Guarded-Based Disjunctive Tuple-Generating Dependencies

A

Guarded-Based Disjunctive Tuple-Generating Dependencies

PIERRE BOURHIS, CNRS CRIStAL, University of Lille 1 and INRIA Lille
MARCO MANNA, University of Calabria
MICHAEL MORAK, Vienna University of Technology
ANDREAS PIERIS, University of Edinburgh

We perform an in-depth complexity analysis of query answering under guarded-based classes of disjunctive tuple-generating dependencies (TGDs), focussing on (unions of) conjunctive queries ((U)CQs). We show that the problem under investigation is very hard, namely 2ExpTime-complete, even for fixed sets of dependencies of a very restricted form. This is a surprising lower bound that demonstrates the enormous impact of disjunction on query answering under guarded-based TGDs, and also reveals the source of complexity for expressive logics such as the guarded fragment of first-order logic. We then proceed to investigate whether prominent subclasses of (U)CQs, i.e., queries of bounded treewidth and hypertree-width, and acyclic queries, have a positive impact on the complexity of the problem under consideration. We show that queries of bounded treewidth and of bounded hypertree-width do not reduce the complexity of our problem, even if we focus on predicates of bounded arity, or on fixed sets of disjunctive TGDs. Regarding acyclic queries, although the problem remains 2ExpTime-complete in general, in some relevant settings the complexity reduces to ExpTime-complete. Finally, with the aim of identifying tractable cases, we focus our attention on atomic queries. We show that atomic queries do not make the query answering problem easier under classes of guarded-based disjunctive TGDs that allow more than one atom to occur in the body of the dependencies. However, the complexity significantly decreases in the case of dependencies that can have only one atom in the body. In particular, we obtain a PTime-completeness if we focus on predicates of bounded arity, and AC_0-membership when the set of dependencies and the query are fixed. Interestingly, our results can be used as a generic tool for establishing complexity results for query answering under various description logics.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems — query processing, relational databases; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving — logic programming, resolution

General Terms: Algorithms, Theory, Languages

Additional Key Words and Phrases: Query answering, conjunctive queries, tuple-generating dependencies, existential rules, disjunction, guardedness, computational complexity

ACM Reference Format:
Pierre Bourhis, Marco Manna, Michael Morak and Andreas Pieris. 2016. Guarded-Based Disjunctive Tuple-Generating Dependencies. *ACM Trans. Datab. Syst.* V, N, Article A (January YYYY), 45 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

Bourhis was partially supported by CPER Nord-Pas de Calais/FEDER DATA Advanced Data Science and Technologies 2015-2020, and the ANR Aggreg Project ANR-14-CE25-0017, INRIA Northern European Associate Team Integrated Linked Data. Manna was partially supported by the Italian Ministry of University and Research, project PON03PE_00001_1, and the Italian Ministry of Economic Development, project F/020016/01-02/X27. Morak was partially supported by the Austrian Science Fund (FWF), project Y698, and the Austrian Academy of Sciences under a DOC Fellowship. Part of the work of Pieris was performed while was a postdoctoral researcher at the Vienna University of Technology; he was supported by the Austrian Science Fund (FWF), projects P25207-N23 and Y698, and the Vienna Science and Technology Fund (WWTF), project ICT12-015.

DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Rule-based languages lie at the core of several areas of central importance to databases and artificial intelligence such as data exchange, deductive databases, and knowledge representation and reasoning, to name a few. An algorithmic task that involves rules and is crucial for the above applications is query answering. The central objective of the current work is to better understand the computational complexity of query answering under a broad class of disjunctive rules.

1.1. Tuple-Generating Dependencies

Tuple-generating dependencies (TGDs) [Beeri and Vardi 1984], a.k.a. as *existential rules* [Baget et al. 2011a] and *Datalog*$^{\pm}$ *rules* [Calì et al. 2010], form a prominent rule-based formalism. TGDs are implications of the form $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \exists \mathbf{Y}(\psi(\mathbf{X}, \mathbf{Y})))$, where φ and ψ are conjunctions of atoms, and they essentially state that some tuples in a relational instance imply the presence of some other tuples in the instance (hence the term "tuple-generating"). TGDs were intended as a unifying framework for the large range of relational dependencies introduced and studied in the 1970s and the 1980s [Abiteboul et al. 1995].

By the late 1980s, a sense of fatigue had settled in concerning the study of database dependencies. In his invited paper at PODS 1987 [Ullman 1987], Jeffrey D. Ullman listed database dependency theory in a section titled "Last Gasps of the Dying Swans", and stated the following: "Perhaps it is time for work in these areas to be driven more by potential applications than by the need to investigate yet another class of dependencies." Interestingly, during the last decade, TGDs have found many uses and applications in different areas of database and AI research. For example, they have been used in formalizing and investigating inter-operability tasks such as data exchange [Fagin et al. 2005]. TGDs have been also used in metadata management tasks, and in particular for formalizing operations on schema mappings [Fagin 2007]. Finally, they have been used for knowledge representation and reasoning purposes, and in fact as an alternative way for modeling ontologies [Calì et al. 2010; Baget et al. 2011a].

1.2. Query Answering

Query answering under TGDs is one of the main algorithmic tasks that involves TGDs, and is of special interest for database and AI applications, such as the ones mentioned above. The importance of query answering under constraints has been recognized both by the database and AI communities, and this is documented by a recent Dagstuhl seminar dedicated to querying and reasoning under expressive constraints[1].

A prominent class of queries, which is of special interest to the current work, are *conjunctive queries (CQs)*. CQs are assertions of the form $\exists \mathbf{X}(\varphi(\mathbf{X}, \mathbf{Y}))$, where φ is a conjunction of atoms — if $\mathbf{Y} = \varnothing$, i.e., there are no free variables, then the above CQ is called Boolean. CQs correspond to the select-project-join fragment of relational algebra, and form one of the most natural and commonly used languages for querying relational databases [Abiteboul et al. 1995].

Given a database D, a set Σ of TGDs, and a Boolean conjunctive query $q = \exists \mathbf{X}(\varphi(\mathbf{X}))$, the problem of query answering is defined as follows: decide whether each model of the logical theory $(D \wedge \Sigma)$, i.e., a relational instance that contains D and satisfies Σ, is also a model of q, written as $D \cup \Sigma \models q$. In case $q = \exists \mathbf{X}(\varphi(\mathbf{X}, \mathbf{Y}))$ with free variables $\mathbf{Y}$, the query answering problem consists in deciding whether a $|\mathbf{Y}|$-tuple of constants $\mathbf{t}$ is an answer to $q_{\mathbf{t}}$ over $(D \wedge \Sigma)$, or, equivalently, whether $D \cup \Sigma \models q_{\mathbf{t}}$, where $q_{\mathbf{t}} = \exists \mathbf{X}(\varphi(\mathbf{X}, \mathbf{t}))$. Unfortunately, CQ answering under TGDs is an undecidable problem [Beeri and Vardi

[1] http://www.dagstuhl.de/14331

1981], even for fixed sets of TGDs [Calì et al. 2013], and singleton sets of TGDs [Baget et al. 2011a]. There has been a recent and increasing focus on the development of (semantic and syntactic) conditions that can be applied on the TGDs in order to guarantee the decidability of CQ answering; see, e.g., [Fagin et al. 2005; Krötzsch and Rudolph 2011; Calì et al. 2012b; Leone et al. 2012; Thomazo et al. 2012].

1.3. What about Disjunctive Rules?

Although several expressive TGD-based formalisms, which guarantee the decidability of CQ answering, have been proposed and investigated over the past years, none of them is powerful enough for nondeterministic guessing, or, simply, for expressing disjunctive TGDs of the form $\forall \mathbf{X} \left(\varphi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}_i \left(\psi_i(\mathbf{X}, \mathbf{Y}_i)\right)\right)$, introduced in [Deutsch and Tannen 2003]. (Notice that by allowing disjunction in the left-hand side of a TGD we do not add expressive power since such a rule can be transformed into an equivalent set of TGDs, just by introducing a TGD for each disjunct.) The usefulness of disjunctive TGDs for database and AI applications is generally acknowledged. In what follows, we focus on three such applications, namely metadata management, database querying, and knowledge representation and reasoning, and we briefly discuss how disjunctive TGDs can be used:

(1) *Quasi-Inverses of Schema Mappings.* Schema mappings are high-level specifications that describe the relationship between two database schemas [Fagin et al. 2005], and form the vital building block of crucial data interoperability tasks such as data exchange. An important metadata management task is schema mapping inversion. This is done by applying an inverse operator that, intuitively speaking, transforms a schema mapping $\mathcal{M}$ into a schema mapping $\mathcal{M}'$ such that, if after applying $\mathcal{M}$ we then apply $\mathcal{M}'$, the resulting effect of $\mathcal{M}'$ is to "cancel" the effect of $\mathcal{M}$, i.e., the composition of $\mathcal{M}$ with $\mathcal{M}'$ gives the identity schema mapping [Fagin 2007]. The inverse operator, as described above, turned out to be rather restrictive, since it is rare that a schema mapping has an inverse, and a more refined operator, called quasi-inverse, was proposed and investigated in [Fagin et al. 2008]. One of the main results of [Fagin et al. 2008] was a characterization of the language needed to express quasi-inverses of schema mappings, which is based on disjunctive TGDs. If a schema mapping specified by source-to-target TGDs is quasi-invertible, then it has a quasi-inverse specified by a set of target-to-source disjunctive TGDs; the following example has been taken from [Fagin et al. 2008].

Example 1.1 (*Quasi-Inverse*). Let $\mathcal{M}$ be the schema mapping specified by

$$\forall X \forall Y (s(X,Y) \rightarrow p(X,Y))$$
$$\forall X \forall Y (t(X,Y) \rightarrow p(X,X)).$$

There is only one possible generator of $p(X,Y)$ if $X \neq Y$, that is, $s(X,Y)$, and this is captured by the TGD

$$\forall X \forall Y (p(X,Y) \wedge \mathit{constant}(X) \wedge \mathit{constant}(Y) \wedge \mathit{neq}(X,Y) \rightarrow s(X,Y)),$$

where $\mathit{constant}(c)$ states that c is a constant, and $\mathit{neq}(c,d)$ states that $c \neq d$. Furthermore, there are two possible generators of $p(X,X)$, namely $s(X,X)$ and $t(X,Y)$, and this is reflected by the disjunctive TGD

$$\forall X (p(X,X) \rightarrow s(X,X) \vee \exists Y (t(X,Y))).$$

In fact, the algorithm for producing quasi-inverses considers all such generators. ■

Apart from computing the quasi-inverse $\mathcal{M}'$, the task of posing a CQ over $\mathcal{M}'$ is also highly relevant for data exchange purposes. In this case, we are actually facing the

problem of CQ answering under disjunctive TGDs, which is defined in the same way as the problem of CQ answering under TGDs.

(2) *Database Queries.* There are natural queries that can be easily expressed using disjunctive TGDs; for example, checking whether a graph is 3-colorable. This problem can be naturally encoded as a CQ answering problem under disjunctive TGDs (even without existentially quantified variables).

Example 1.2 (*3-Colorability*). Consider a graph $G = (V, E)$, and let D_G be the database that stores G in the natural way, i.e., we have an atom $vertex(v)$ for each $v \in V$, and an atom $edge(v, u)$ for each edge $(v, u) \in E$. We define Σ as follows:

$$\begin{aligned}
\forall X(vertex(X) &\rightarrow red(X) \vee green(X) \vee blue(X)) \\
\forall X \forall Y(edge(X,Y) \wedge red(X) \wedge red(Y) &\rightarrow notLegal) \\
\forall X \forall Y(edge(X,Y) \wedge green(X) \wedge green(Y) &\rightarrow notLegal) \\
\forall X \forall Y(edge(X,Y) \wedge blue(X) \wedge blue(Y) &\rightarrow notLegal).
\end{aligned}$$

Roughly speaking, the first rule assigns to each vertex of G one of the three colors. If the assignment is not a legal coloring, then the atom $notLegal$ is inferred by one of the subsequent rules. It is easy to see that G is 3-colorable iff $D \cup \Sigma \not\models notLegal$. ■

The set of disjunctive TGDs in the above example is actually a disjunctive Datalog program. Datalog is a famous database query language that uses disjunction- and function-free logic programs; see, e.g., [Ceri et al. 1990; Abiteboul et al. 1995]. Disjunctive Datalog, that is, Datalog extended with disjunction, has been thoroughly investigated in [Eiter et al. 1997], where it was convincingly argued that disjunction is very useful, and actually necessary, for expressing some practical problems, such as the one in Example 1.2. Disjunctive TGDs extend disjunctive Datalog with existential quantification in the right-hand side of the rules, in the same way as TGDs extend (plain) Datalog with existential quantification.

(3) *Knowledge Representation and Reasoning. Description Logics (DLs)* are languages for knowledge representation and reasoning, which are based on concepts (classes of objects) and roles (binary relations on objects). DLs are amongst the most popular knowledge representation formalisms, and they have been employed in several application areas, such as health and life sciences, natural language processing, and data integration, to name a few; for more applications and details see [Baader et al. 2003]. They also play a crucial role in the Semantic Web, where they provide the logical underpinning for the OWL language, the standard way for modeling Semantic Web ontologies [Cuenca Grau et al. 2008]. It is interesting to observe that several key DLs are captured by disjunctive TGDs.

Example 1.3 (*Description Logics*). Consider the following axioms expressed in $\mathcal{ALC}$, one of the central DLs, introduced in [Schmidt-Schauß and Smolka 1991], which is at the basis of many other expressive DLs:

$$\begin{aligned}
\exists parentOf.isparent &\sqsubseteq \exists grandparentOf.human \\
human &\sqsubseteq male \sqcup female.
\end{aligned}$$

The former states that "each parent of a parent is a grandparent of a human", while the latter states that "each human is a male or a female". It is easy to verify that the same can be expressed using the following disjunctive TGDs:

$$\begin{aligned}
\forall X \forall Y(parentOf(X,Y) \wedge isparent(Y) &\rightarrow \exists Z(grandparentOf(X,Z) \wedge human(Z))) \\
\forall X(human(X) &\rightarrow male(X) \vee female(X)).
\end{aligned}$$

Notice that for expressing DL axioms as disjunctive TGDs, we just need unary predicates for concepts, and binary predicates for roles. ■

A large corpus of work on DLs has focused on the problems of consistency, instance checking and logical entailment. However, the last few years, the attention has shifted to the problem of CQ answering (see, e.g., [Rudolph and Glimm 2010; Eiter et al. 2012; Calvanese et al. 2013] and references therein), and thus, once again, we are facing the problem of answering CQs under disjunctive TGDs.

1.4. Guardedness and Disjunctive TGDs

It is evident that query answering under disjunctive TGDs is a crucial algorithmic task with several applications. However, as mentioned above, CQ answering under TGDs is already an undecidable problem [Beeri and Vardi 1984]. A key decidability paradigm, which is crucial for the current work, is *guardedness*, a well-known restriction that guarantees good model-theoretic and computational properties for first-order sentences. More precisely, the *guarded fragment of first-order logic (GFO)* was introduced in [Andréka et al. 1998] with the aim of explaining and generalizing the good properties of modal logic. Guarded formulas are constructed as usual first-order formulas with the exception that all quantification must be guarded, i.e., of the form $\forall \mathbf{X}(\underline{a} \rightarrow \varphi)$ and $\exists \mathbf{X}(\underline{a} \wedge \varphi)$, where $\underline{a}$ is an atomic formula that guards φ in the sense that it contains all the free variables of φ. It has been shown that modal logic can be embedded in GFO, and argued in a convincing way that GFO inherits the good properties of modal logic. Furthermore, in [Grädel 1999], it has been established that GFO enjoys the tree-model property, i.e., if a GFO sentence admits a model, then it admits a tree-like model, which in turn implies the decidability of the main reasoning tasks, i.e., satisfiability and query answering. Recently, inspired by GFO, the class of guarded TGDs, that is, TGDs with a guard-atom in the left-hand side that contains (or guards) all the universally quantified variables, has been proposed and investigated [Calì et al. 2013]. Several interesting extensions and restrictions of guarded TGDs have also been considered [Baget et al. 2011a; Calì et al. 2012a]; we refer to all those formalisms by the term guarded-based TGDs.

Guarded-based TGDs can be naturally extended to disjunctive TGDs, without losing the good model-theoretic properties and the decidability of CQ answering. Moreover, the obtained formalisms, apart from the fact that they are theoretically interesting, have several practical applications — observe that all the disjunctive TGDs employed in Examples 1.1, 1.2 and 1.3 are guarded. In fact, it is not difficult to show that, if a schema mapping specified by source-to-target GAV (global-as-view) TGDs, i.e., TGDs with just one atom in the right-hand side, is quasi-invertible, then it has a quasi-inverse specified by a set of target-to-source guarded disjunctive TGDs; this follows by the definition of the algorithm for producing quasi-inverses given in [Fagin et al. 2008]. Moreover, it is possible to show that the core DL $\mathcal{ALC}$ (end some extensions of it) is captured by guarded disjunctive TGDs; more details are given in Section 8. From the above discussion, we see that guarded-based disjunctive TGDs form a family of first class formalisms with several applications, and thus they deserve our attention.

Although it was known that the problem of CQ answering under guarded-based disjunctive TGDs is decidable (implicit in [Calì et al. 2008]), little was known about its computational complexity before the conference papers [Gottlob et al. 2012; Bourhis et al. 2013; 2014], which are significantly extended and improved by the present journal paper. An exception is the work [Alviano et al. 2012], where some special cases of the problem are considered — more details about the complexity results that are inherited from [Alviano et al. 2012] are given in the technical sections.

1.5. Our Main Goal

The main goal of the current work is to better understand the impact of disjunction on query answering under the main guarded-based classes of TGDs. Towards this direction, we perform an in-depth complexity analysis along three different dimensions of the problem under consideration:

(1) *Complexity Type.* Following Vardi's taxonomy [Vardi 1982], the data complexity of our problem is calculated taking only the database as input, while the query and the set of dependencies are considered fixed. The combined complexity is the complexity calculated considering as input, together with the database, also the query and the set of dependencies. Apart from the combined complexity, we would also like to understand how the complexity of our problem is affected when some key parameters are fixed. In particular, we consider the following two variants of the combined complexity: (1) the bounded-arity combined complexity, which is calculated by assuming that the arity of the underlying schema is bounded; and (2) the fixed-program combined complexity, which is calculated by considering the set of disjunctive TGDs as fixed (the set of dependencies is usually called program, and hence the term "fixed-program"). Notice that, in practice, the arity of the underlying schema is usually small and can be effectively assumed to be fixed. Moreover, the components that change quite often over time are the database and the query, while the program remains the same.

(2) *Dependency Language.* As already said, we are going to focus on the main guarded-based classes of TGDs extended with disjunction. More precisely, we are going to consider the classes of guarded and frontier-guarded disjunctive TGDs. Recall that guarded disjunctive TGDs have a guard atom in the left-hand side that contains all the universally quantified variables [Calì et al. 2013]. Frontier-guarded disjunctive TGDs extend guarded disjunctive TGDs by requiring in the guard atom only the universally quantified variables that appear also in the right-hand side of the dependency [Baget et al. 2011a]. Each one of the above classes has its weakly counterpart that we would also like to consider in our investigation. Two lightweight formalisms that also deserve our attention are disjunctive inclusion dependencies (IDs), and linear disjunctive TGDs. A detailed exposition of all considered formalisms can be found in Section 3.

(3) *Query Language.* The main query language considered in the current study is the language of CQs. A natural extension of CQs, which has been heavily studied in the past, and will also be part of our complexity analysis, is the class of *union of conjunctive queries (UCQs)*; see, e.g., [Abiteboul et al. 1995]. Several subclasses of (U)CQs have been also considered in the literature with the aim of reducing the complexity of classical database problems such as query evaluation and query containment. The above problems are NP-hard, in general, and become tractable if restricted to one of the following languages: (U)CQs of bounded treewidth [Chekuri and Rajaraman 2000], (U)CQs of bounded hypertree-width [Gottlob et al. 2002], acyclic (U)CQs [Chekuri and Rajaraman 2000], and atomic queries. We would like to understand whether the above subclasses of (U)CQs have an analogous positive impact on query answering under guarded-based disjunctive TGDs.

1.6. Technical Challenges

The problem of (U)CQ answering under disjunctive TGDs adheres to the standard logical semantics of entailment ($\models$), which denotes entailment under arbitrary, not necessarily finite, models. This implies that, in general, a database and a set of disjunctive TGDs admit infinitely many models, where each of them can be of infinite size; in fact,

this holds already for TGDs due to the existentially quantified variables. Interestingly, in the case of TGDs, it is always possible to construct the so-called *universal* (a.k.a. *canonical*) model, which can be seen as a representative of all the other models. Formally, a model of a logical theory is universal if it can be homomorphically embedded into every other model of the theory. Such a universal model of a database and a set of TGDs can be constructed by applying a well-known procedure called the *chase* (see, e.g., [Johnson and Klug 1984; Fagin et al. 2005; Deutsch et al. 2008]). Roughly, the chase adds new atoms to the given database as dictated by the given TGDs, possibly involving labeled null values as witnesses for the existentially quantified variables, until the final result satisfies all the TGDs. Therefore, for query answering purposes, instead of considering all the models of a database and a set of TGDs, one can focus on the (possibly infinite) universal model.

The situation changes dramatically if we consider disjunctive TGDs. In fact, there is no single model anymore that acts as a representative of all the other models. However, the notion of the universal model can be naturally extended to the notion of the *universal model set*. Formally, given a database D and a set Σ of disjunctive TGDs, a set S of models of $(D \wedge \Sigma)$ is called universal if, for each model M of $(D \wedge \Sigma)$, there exists a model in S that can be homomorphically embedded into M. Such a universal model set of a database and a set of disjunctive TGDs can be constructed by applying the so-called *disjunctive chase*, which is a natural extension of the chase procedure introduced in [Deutsch and Tannen 2003]. However, a universal model set is, in general, infinite, i.e., consists of infinitely many models of infinite size. This is a strong sign that the problem of (U)CQ answering under disjunctive TGDs is more challenging than (U)CQ answering under TGDs, and novel algorithmic techniques must be devised.

1.7. Summary of Contributions

After some technical definitions and preliminaries in Section 2, and the formal definitions of the main guarded-based classes of disjunctive TGDs in Section 3, we proceed with our complexity analysis. In Section 4, we focus on arbitrary (U)CQs, in Section 5 on (U)CQs of bounded treewidth and bounded hypertree-width, in Section 6 on acyclic (U)CQs, and finally, in Section 7, on atomic queries. Each one of the above technical sections starts with an overview subsection, which presents a summary of the obtained results, together with a brief description of the employed techniques, and discusses results inherited from existing works. In what follows, we summarize the general findings of our investigation, and highlight the main results of our complexity analysis:

(1) *Arbitrary (U)CQs.* For the expressive formalisms, i.e., (weakly-)(frontier-)guarded TGDs, the addition of disjunction does not have a significant effect on the complexity of (U)CQ answering. However, the main result of Section 4 (Theorem 4.8) shows that for the less expressive languages, i.e., linear TGDs and inclusion dependencies, the impact can be enormous with, e.g., a jump from NP-completeness to 2ExpTime-completeness in the case of a fixed set of disjunctive IDs; the latter holds even if we focus on predicates of arity at most three.

(2) *(U)CQs of Bounded Treewidth and Hypertree-width.* The main result of Section 5 (Theorem 5.1) shows that queries of bounded treewidth and hypertree-width behave in the same way as arbitrary (U)CQs, and thus they do not have the expected positive impact on the problem under consideration.

(3) *Acyclic (U)CQs.* The main results of Section 6 (Theorems 6.2 and 6.3) show that for acyclic (U)CQs a positive impact can be observed on some relevant settings of the problem. More precisely, for (weakly-)guarded disjunctive TGDs the complexity is reduced from 2ExpTime-complete to ExpTime-complete when we focus on rules

with bounded number of body-variables and predicates of bounded arity, while for (weakly-)frontier-guarded disjunctive TGDs the same can be observed if we focus on fixed programs.

(4) *Atomic Queries.* Finally, we observe that atomic queries do not make our problem easier under the classes of disjunctive TGDs that allow more than one atom in the left-hand side of the rules, i.e., (weakly-)(frontier-)guarded disjunctive TGDs. However, the complexity in the case of disjunctive IDs and linear disjunctive TGDs significantly decreases. In particular, the main results of Section 7 (Theorems 7.11 and 7.13) show a PTIME-membership if we focus on predicates of bounded arity, and AC_0-membership in data complexity; notice that, for atomic queries, the fixed-program combined complexity and the data complexity coincide.

Interestingly, our results provide refined lower bounds for the problem of querying the guarded fragment of first-order logic. Moreover, our techniques and results can be used as a generic tool for establishing results on query answering under several central DLs; details are given in Section 8.

2. PRELIMINARIES

In this section, we present background material necessary for this paper. We recall some basics on relational databases, (Boolean) conjunctive queries, and disjunctive tuple-generating dependencies. We also introduce the technical tools that we are going to employ in our proofs, namely the disjunctive chase procedure, guarded-based fragments of first-order logic, and some basics on alternating Turing machines. We assume the reader is familiar with fundamental concepts of complexity theory. A detailed exposition of all complexity notions employed in this work can be found in the standard textbooks (see, e.g., [Papadimitriou 1994]).

2.1. General

Let $\mathbf{C}$, $\mathbf{N}$ and $\mathbf{V}$ be pairwise disjoint infinite countably sets. The elements of $\mathbf{C}$ are called *constants* (constitute the normal domain of a database), the elements of $\mathbf{N}$ are called *(labeled) nulls* (used as placeholders for unknown values, and thus can be also seen as (globally) existentially quantified variables), and the elements of $\mathbf{V}$ are called *(regular) variables* (used in queries and dependencies). A fixed lexicographic order is assumed on $\mathbf{C} \cup \mathbf{N}$, such that every value in $\mathbf{N}$ follows all those in $\mathbf{C}$. We denote by $\mathbf{X}$ sequences (or sets) of variables $X_1, \ldots, X_k$. Let $[n] = \{1, \ldots, n\}$, for every $n \geqslant 1$.

A *term* t is a constant ($t \in \mathbf{C}$), labeled null ($t \in \mathbf{N}$), or variable ($t \in \mathbf{V}$). An *atomic formula* (or simply *atom*) has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. For an atom $\underline{a}$, we denote by $dom(\underline{a})$ and $var(\underline{a})$ the sets of its terms and variables, respectively. These notations naturally extend to sets of atoms. For convenience, usually conjunctions and disjunctions of atoms are treated as sets of atoms. An *instance* I is a (possibly infinite) set of atoms of the form $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of constants and labeled nulls. A *database* D is a finite instance where only constants occur. Whenever an instance I is treated as a logical formula, it is in fact the formula $\exists \mathbf{X} \, (\bigwedge_{\underline{a} \in I} I)$, where $\mathbf{X}$ contains a variable X_z, for each null z in I.

A *substitution* from a set of symbols S to a set of symbols S' is a partial function $h : S \to S'$ defined as follows: $\varnothing$ is a substitution (empty substitution), and if h is a substitution, then $h \cup \{s \to s'\}$, where $s \in S$, $s' \in S'$ and $h(s)$ is undefined, is a substitution. The *restriction* of h to $T \subseteq S$, denoted $h_{|T}$, is the substitution $h' = \{t \to h(t) \mid t \in T\}$. A *homomorphism* from a set of atoms A to a set of atoms A' is a substitution $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ such that: *(i)* if $t \in \mathbf{C}$, then $h(t) = t$; and *(ii)* if $p(t_1, \ldots, t_n) \in A$, then $h(p(t_1, \ldots, t_n)) = p(h(t_1), \ldots, h(t_n)) \in A'$.

2.2. Queries

A *conjunctive query (CQ)* q is a first-order formula $\exists \mathbf{Y}(\varphi(\mathbf{X}, \mathbf{Y}))$, where φ is a conjunction of atoms with variables from $\mathbf{X} \cup \mathbf{Y} \subset \mathbf{V}$, and possibly constants of $\mathbf{C}$. The *arity* of q is defined as the cardinality of $\mathbf{X}$, i.e., the number of free variables occurring in q. A 0-ary CQ is called *Boolean CQ (BCQ)*. An n-ary *union of conjunctive queries (UCQ)* is a disjunction of a finite number of n-ary CQs. By abuse of notation, sometimes we consider a UCQ as set of CQs. The *answer* to an n-ary CQ $q = \exists \mathbf{Y}(\varphi(\mathbf{X}, \mathbf{Y}))$ over an instance I, denoted $q(I)$, is the set of all tuples of constants $\mathbf{t} \in \mathbf{C}^n$ for which there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple as possible answer, in which case it is said to have a positive answer. Formally, a BCQ has a *positive* answer over I, written as $I \models q$, if $q(I) \neq \varnothing$. The answer to an n-ary UCQ Q over an instance I, denoted $Q(I)$, is the set of n-tuples $\bigcup_{q \in Q} q(I)$. The answer to a union of BCQs over I is positive, written as $I \models Q$, if $Q(I) \neq \varnothing$.

2.3. Disjunctive Tuple-generating Dependencies

A *disjunctive tuple-generating dependency (DTGD)* σ is a first-order formula

$$\forall \mathbf{X} \left(\varphi(\mathbf{X}) \ \rightarrow \ \bigvee_{i=1}^{n} \exists \mathbf{Y}_i \left(\psi_i(\mathbf{X}, \mathbf{Y}_i) \right) \right),$$

where $n \geqslant 1$, $\mathbf{X} \cup \mathbf{Y}_1 \cup \ldots \cup \mathbf{Y}_n \subset \mathbf{V}$, and $\varphi, \psi_1, \ldots, \psi_n$ are conjunctions of atoms (possibly with constants of $\mathbf{C}$). The formula φ is called the *body* of σ, denoted $body(\sigma)$, while $\bigvee_{i=1}^{n} \psi_i$ is the *head* of σ, denoted $head(\sigma)$. The set of variables $var(body(\sigma)) \cap var(head(\sigma)) \subseteq \mathbf{X}$, i.e., the variables of $\mathbf{X}$ which appear both in the body and in the head of σ, is known as the *frontier* of σ, and is denoted as $frontier(\sigma)$. If $n = 1$, then σ is called a *tuple-generating dependency (TGD)*. The *schema* of a set Σ of DTGDs, denoted $sch(\Sigma)$, is the set of all predicates occurring in Σ. In the rest of the paper, for brevity, we will omit the universal quantifiers in front of DTGDs, and implicitly assume such a quantification. We will also use the comma (instead of $\wedge$) for conjoining atoms in the body and in the head of a DTGD, and omit the parentheses whenever the scope of an existential quantifier is clear. An instance I satisfies σ, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, then there exists $i \in [n]$ and $h' \supseteq h$ such that $h'(\psi_i(\mathbf{X}, \mathbf{Y}_i)) \subseteq I$; I satisfies a set Σ of DTGDs, denoted $I \models \Sigma$, if $I \models \sigma$, for each $\sigma \in \Sigma$. Whenever a set Σ of DTGDs is treated as a logical formula, it is in fact the formula $(\bigwedge_{\sigma \in \Sigma} \sigma)$.

2.4. Query Answering

Given a database D and a set Σ of DTGDs, the answers we consider are those that are true in *all* models of D with respect to Σ. Formally, the *models* of D with respect to Σ, denoted as $mods(D, \Sigma)$, is the set of all instances I such that $I \supseteq D$ and $I \models \Sigma$. The *answer* to an n-ary CQ q w.r.t. D and Σ is the set of n-tuples of constants

$$ans(q, D, \Sigma) \ = \ \bigcap_{I \in mods(D, \Sigma)} q(I).$$

If q is Boolean, then its answer is *positive*, denoted $D \cup \Sigma \models q$, if $ans(q, D, \Sigma) \neq \varnothing$. The answer to a UCQ w.r.t. D and Σ is defined analogously. The main decision problems tackled in this work are defined as follows:

Definition 2.1 *((U)CQ Answering)*. Given a database D, a set Σ of DTGDs, an n-ary CQ q, and a tuple $\mathbf{t} \in \mathbf{C}^n$, *CQ answering* is the problem of deciding whether $\mathbf{t} \in ans(q, D, \Sigma)$. The *UCQ answering* problem is defined analogously. □

It is well-known that CQ answering for arbitrary CQs can be easily reduced to CQ answering for BCQs, just by substituting the given tuple $\mathbf{t}$ into the CQ; thus, we can focus on BCQs. Henceforth, by CQ and UCQ we refer to a BCQ and a union of BCQs, respectively. The *data complexity* of the above problems is computed taking only the database as input. For the *combined complexity*, the query and set of DTGDs count as part of the input as well. We assume that the database and the query use only predicates of $sch(\Sigma)$. Let us explain why this assumption can be made without affecting the generality of our complexity results. Consider a database D, a set Σ of DTGDs, and a CQ q. Let $D = D' \cup D''$, where $\{D', D''\}$ is a partition of D and $D' = \{p(\mathbf{t}) \in D \mid p \in sch(\Sigma)\}$. Moreover, let $q' = \exists \mathbf{X}(\varphi(\mathbf{X}) \wedge \psi(\mathbf{X}))$, where, for each predicate p occurring in $\varphi(\mathbf{X})$ (resp., $\psi(\mathbf{X})$), $p \in sch(\Sigma)$ (resp., $p \notin sch(\Sigma)$). Clearly, if there exists a homomorphism h such that $h(\psi(\mathbf{X})) \subseteq D''$, then $D \cup \Sigma \models q$ iff $D' \cup \Sigma \models \exists \mathbf{X} h(\varphi(\mathbf{X}))$; otherwise, $D \cup \Sigma \not\models q$. Notice that CQ answering is at least as hard as checking for the existence of h. The above argument can be straightforwardly extended to UCQs.

2.5. Normal Form of DTGDs

For query answering purposes, we can focus on DTGDs that are in normal form. A set Σ of DTGDs is in normal form if each $\sigma \in \Sigma$ is of the form

$$\varphi(\mathbf{X}) \rightarrow p(\mathbf{X}) \qquad \text{or} \qquad \varphi(\mathbf{X}) \rightarrow \exists Y\, p(\mathbf{X}, Y) \qquad \text{or} \qquad \varphi(\mathbf{X}) \rightarrow p_1(\mathbf{X}) \vee p_2(\mathbf{X}).$$

The first refers to existential-free single-atom-head TGDs, the second to single-atom-head TGDs with exactly one occurrence of an existentially quantified variable, while the third refers to existential-free DTGDs with a disjunction of two atoms in the head. Every set Σ of DTGDs can be transformed in logarithmic space into a set $\mathsf{N}(\Sigma)$ in normal form such that, for every database D and UCQ Q, $D \cup \Sigma \models Q$ iff $D \cup \mathsf{N}(\Sigma) \models Q$. The normalization procedure can be found in Appendix A.

In the sequel, in order to simplify our technical proofs, sometimes we will focus on normalized sets of DTGDs. It will be explicitly stated when the DTGDs under consideration are assumed to be in normal form.

2.6. Disjunctive Chase

In our investigation of (U)CQ answering, we employ the *disjunctive chase* introduced in [Deutsch and Tannen 2003], an extension of the well-known chase procedure. Each disjunctive chase step "branches" out several instances, each satisfying one of the disjuncts of the DTGD that is applied, and thus the result of the disjunctive chase is, in general, a set of instances (and not a single instance as in the classical chase). The disjunctive chase works on an instance through the so-called DTGD chase rule:

Definition 2.2 (*Chase Rule*). Consider an instance I, and a DTGD σ of the form $\varphi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}\, \psi_i(\mathbf{X}, \mathbf{Y})$. σ is *applicable* to I if there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq I$, and the result of applying σ to I with h is the set $\{I_1, \ldots, I_n\}$, where $I_i = I \cup h'(\psi_i(\mathbf{X}, \mathbf{Y}))$, for each $i \in [n]$, and $h' \supseteq h$ is such that $h'(Y)$ is a "fresh" null not occurring in I, and following lexicographically all those in I, for each $Y \in \mathbf{Y}$. For such an application, which defines a single *chase step*, we write $I\langle\sigma, h\rangle\{I_1, \ldots, I_n\}$. □

A *disjunctive chase tree* of a database D and a set Σ of DTGDs is a (possibly infinite) tree such that the root is D, and for every node I, assuming that $\{I_1, \ldots, I_n\}$ are the children of I, there exists $\sigma \in \Sigma$ and a homomorphism h such that $I\langle\sigma, h\rangle\{I_1, \ldots, I_n\}$. The disjunctive chase algorithm for D and Σ consists of an exhaustive application of DTGD chase steps in a *fair fashion*, which leads to a disjunctive chase tree T of D and Σ; let $chase(D, \Sigma)$ be the set $\{I \mid I \text{ is a leaf of } T\}$. Notice that each leaf of T is well-defined as the least fixpoint of a monotonic operator. By construction, each instance of

$chase(D, \Sigma)$ is a model of D and Σ. Interestingly, $chase(D, \Sigma)$ is a *universal model set* of D and Σ, i.e., for each $I \in mods(D, \Sigma)$, there exists $J \in chase(D, \Sigma)$ and a homomorphism h_J such that $h_J(J) \subseteq I$ [Deutsch et al. 2008]. This universality property implies the following useful result, which actually shows that the disjunctive chase is a formal algorithmic tool for query answering purposes.

THEOREM 2.3. *Consider a database D, a set Σ of DTGDs, and a UCQ Q. It holds that $D \cup \Sigma \models Q$ iff $I \models Q$, for each $I \in chase(D, \Sigma)$.*

2.7. Guarded-based Fragments of First-order Logic

In our complexity analysis, we will exploit complexity results established for expressive guarded-based fragments of first-order logic.

Guarded Fragment. The *guarded fragment of first-order logic (GFO)*, introduced in [Andréka et al. 1998], is the collection of first-order formulas with some syntactic restrictions on quantification patterns. The set of GFO formulas over a schema $\mathcal{R}$ is the smallest set

(1) containing all atomic formulas using predicates from $\mathcal{R}$ and equalities;
(2) closed under the logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$; and
(3) if $\underline{a}$ is an atom or an equality containing all the variables of $\mathbf{X} \cup \mathbf{Y}$, and φ is a GFO formula with free variables contained in $(\mathbf{X} \cup \mathbf{Y})$, then

$$\forall \mathbf{X}(\underline{a} \rightarrow \varphi) \qquad \text{and} \qquad \exists \mathbf{X}(\underline{a} \wedge \varphi)$$

are GFO formulas as well.

Guarded Negation. *Guarded negation first-order logic (GNFO)* restricts first-order logic by requiring that all occurrences of negation are of the form $\underline{a} \wedge \neg\varphi$, where $\underline{a}$ is an atom containing all the free variables of φ [Bárány et al. 2015]. Formally, the formulas of GNFO are generated by the recursive definition

$$\varphi \ ::= \ p(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists X\, \varphi \mid \underline{a} \wedge \neg\varphi,$$

where each $t_i \in \mathbf{C} \cup \mathbf{V}$, and in the last clause, $\underline{a}$ is an atomic formula containing all free variables of φ. Notice that GNFO is strictly more expressive than GFO.

2.8. Alternating Turing Machines

An *alternating Turing machine* is a tuple $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, Λ is the tape alphabet, $\delta \subseteq (S \times \Lambda) \times (S \times \Lambda \times \{-1, +1\})$ is the transition relation, and $s_0 \in S$ is the initial state. We assume that Λ contains a special blank symbol $\sqcup$. The symbols -1 and $+1$ denote the cursor directions left and right, respectively.

A *computation tree* for M is a tree labeled by configurations (tape content, cursor position, and internal state) of M such that (1) if node v is labeled by an existential configuration, then v has one child, labeled by one of the possible successor configurations; (2) if v is labeled by a universal configuration, then v has one child for each possible successor configuration; (3) the root is labeled by the initial configuration; and (4) all leaves are labeled by accepting or rejecting configurations. A computation tree is *accepting* if it is finite and all leaves are labeled by accepting configurations.

3. GUARDED-BASED CLASSES OF DTGDS

(U)CQ answering under (non-disjunctive) TGDs is undecidable [Beeri and Vardi 1981], even when the set of TGDs is fixed [Calì et al. 2013], or even when the set of TGDs is a

singleton [Baget et al. 2011a]. Several syntactic restrictions can be found in the literature that guarantee the decidability of (U)CQ answering under TGDs. The property which is of special interest for the current work is *guardedness*. In particular, guardedness ensures that the instance constructed by the chase procedure is tree-like, which in turn implies the decidability of (U)CQ answering. In this section, we introduce the disjunctive version of the main guarded-based classed of TGDs that can be found in the literature. Then, we show that the tree-likeness of the chase is preserved, and thus, as we shall see, also the decidability of the problems under consideration is preserved.

Before we proceed further, let us recall the notion of *affected positions*, introduced in [Calì et al. 2013]. Given an n-ary predicate p, the position $p[i]$ is identified by p and its i-th attribute. We refer to the arity of p by $\mathit{arity}(p)$. Given a set Σ of DTGDs, the set of positions of $\mathit{sch}(\Sigma)$, denoted $\mathit{pos}(\Sigma)$, is the set $\{p[i] \mid p \in \mathit{sch}(\Sigma) \text{ and } i \in [\mathit{arity}(p)]\}$. Intuitively, a position $\pi \in \mathit{pos}(\Sigma)$ is affected if, during the construction of the disjunctive chase w.r.t. Σ, π may host a null value. Formally, the set of affected positions of Σ, denoted $\mathit{affected}(\Sigma)$, is inductively defined as follows: (1) if there exists $\sigma \in \Sigma$ such that an existentially quantified variable occurs at position π of $\mathit{pos}(\Sigma)$, then $\pi \in \mathit{affected}(\Sigma)$; and (2) if there exists $\sigma \in \Sigma$ and a variable X that occurs in $\mathit{body}(\sigma)$ only at positions of $\mathit{affected}(\Sigma)$, and X appears in $\mathit{head}(\sigma)$ at position π, then $\pi \in \mathit{affected}(\Sigma)$.

Example 3.1 (*Affected Positions*). Consider the set Σ consisting of the DTGDs:

$$\begin{aligned}\sigma_1 &= t(X,Y), s(Y), s(Z) \rightarrow \exists W\, t(Y,W)\\ \sigma_2 &= t(X,Y) \rightarrow p(Y)\\ \sigma_3 &= p(X) \rightarrow p_1(X) \vee p_2(X).\end{aligned}$$

Clearly, $t[2] \in \mathit{affected}(\Sigma)$ due to the existentially quantified variable occurring at $t[2]$ in $\mathit{head}(\sigma_1)$. Since the variable Y occurs at position $t[2]$ in $\mathit{body}(\sigma_2)$ and at position $p[1]$ in $\mathit{head}(\sigma_2)$, we conclude that $p[1] \in \mathit{affected}(\Sigma)$. Finally, due to σ_3, both $p_1[1]$ and $p_2[1]$ belong to $\mathit{affected}(\Sigma)$. The position $t[1]$ is not affected w.r.t. Σ since, although the same variable Y occurs at $t[2] \in \mathit{affected}(\Sigma)$ in $\mathit{body}(\sigma_1)$ and at $t[1]$ in $\mathit{head}(\sigma_1)$, Y occurs also at the non-affected position $s[1]$. ■

Having the above auxiliary notion in place, we are now ready to define the main guarded-based classes of DTGDs.

Definition 3.2 (*Classes of DTGDs*). Fix a DTGD σ.

— σ is *weakly-frontier-guarded* w.r.t. a set Σ of DTGDs if there exists an atom $\underline{a} \in \mathit{body}(\sigma)$ which contains (or guards) all the variables of $\mathit{frontier}(\sigma)$ that appear only at positions of $\mathit{affected}(\Sigma)$ — the set Σ is called weakly-frontier-guarded if, for every $\sigma' \in \Sigma$, σ' is weakly-frontier-guarded w.r.t. Σ;
— σ is *weakly-guarded* w.r.t. a set Σ of DTGDs if there exists an atom $\underline{a} \in \mathit{body}(\sigma)$ which contains (or guards) all the variables of $\mathit{var}(\mathit{body}(\sigma))$ that appear only at positions of $\mathit{affected}(\Sigma)$ — the set Σ is called weakly-guarded if, for every $\sigma' \in \Sigma$, σ' is weakly-guarded w.r.t. Σ;
— σ is *frontier-guarded* (resp., *guarded*) if there exists an atom $\underline{a} \in \mathit{body}(\sigma)$ which contains all the variables of $\mathit{frontier}(\sigma)$ (resp., $\mathit{var}(\mathit{body}(\sigma))$);
— σ is *multi-linear* if every atom $\underline{a} \in \mathit{body}(\sigma)$ contains all the variables of $\mathit{var}(\mathit{body}(\sigma))$;
— σ is *linear* if $|\mathit{body}(\sigma)| = 1$, i.e., σ has only one body-atom; and
— σ is a *disjunctive inclusion dependency (DID)* if it is constant-free, $|\mathit{body}(\sigma)| = 1$, $\mathit{head}(\sigma)$ is a disjunction of atoms, each $X \in \mathit{var}(\mathit{body}(\sigma))$ occurs at most once in $\mathit{body}(\sigma)$, and for each atom $\underline{a} \in \mathit{head}(\sigma)$, each $X \in \mathit{var}(\underline{a})$ occurs at most once in $\underline{a}$. □

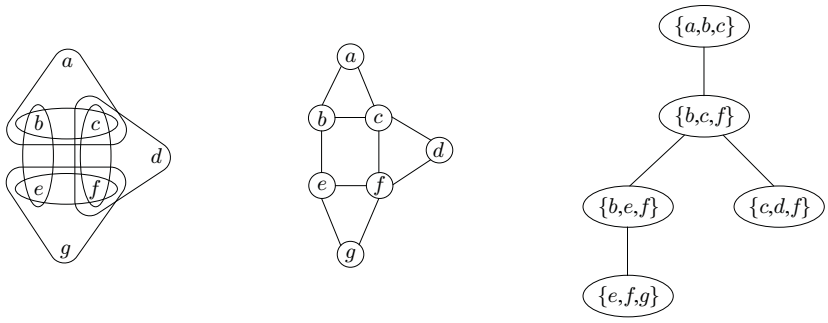

Fig. 1: (a) The hypergraph $\mathcal{H}(I)$; (b) the Gaifman graph $G_{\mathcal{H}(I)}$; and (c) a tree decomposition of $G_{\mathcal{H}(I)}$.

Notice that the above classes of DTGDs are closed under the normalization procedure introduced in Section 2, i.e., for a set Σ of DTGDs which falls in a class $\mathcal{C}$, where $\mathcal{C}$ is one of the classes introduced in Definition 3.2, $\mathsf{N}(\Sigma)$ also falls in $\mathcal{C}$.

Decidability of UCQ Answering. We conclude this section by showing that indeed UCQ answering under weakly-frontier-guarded sets of DTGDs (and thus, under all the other classes of DTGDs introduced above) is decidable. We show this by exploiting the classical result that fragments of first-order logic which enjoy the *finite treewidth model property* are decidable [Courcelle 1989][2]. The treewidth of a graph is a well-known notion in graph theory, which measures how similar the graph is to a tree.

Definition 3.3 (*Treewdith*). Given a graph $G = (V, E)$, a *tree decomposition* of G is a pair (T, λ), where $T = (N, A)$ is a tree, and λ is a labeling function $N \to 2^V$ such that:

(1) for all $v \in V$, there exists $u \in N$ such that $v \in \lambda(u)$;
(2) for all edges $(v_1, v_2) \in E$, there exists $u \in N$ such that $\{v_1, v_2\} \subseteq \lambda(u)$; and
(3) for every $v \in V$, the set $\{u \mid v \in \lambda(u)\} \subseteq N$ induces a connected subtree of T.

The *width* of (T, λ) is $\max_{u \in N}\{|\lambda(u)|\} - 1$. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width of all possible tree decompositions of G. □

Let us now recall the treewidth of an instance I. The hypergraph of I, denote $\mathcal{H}(I)$, is a hypergraph (V, H), where $V = dom(I)$, and, for each $\underline{a} \in I$, there exists a hyperedge $h \in H$ such that $h = dom(\underline{a})$. The Gaifman graph of $\mathcal{H}(I)$ is the graph $G_{\mathcal{H}(I)} = (V, E)$, where V is the node set of $\mathcal{H}(I)$, and $(v, u) \in E$ iff $\mathcal{H}(I)$ has a hyperedge h such that $\{v, u\} \subseteq h$. The treewidth of an instance I, denoted $tw(I)$, is defined as the treewidth of its hypergraph $\mathcal{H}(I)$, which in turn is the treewidth of $G_{\mathcal{H}(I)}$.

Example 3.4 (*Treewidth of an Instance*). Consider the instance

$$I \;=\; \{p(a,b,c), p(c,f,d), p(e,f,g), t(b,c), t(b,e), t(e,f), t(f,c)\},$$

where $\{a, b, c, d, e, f, g\} \subset \mathbf{C}$. The hypergraph $\mathcal{H}(I)$ is depicted in Figure 1(a), its Gaifman graph $G_{\mathcal{H}(I)}$ in Figure 1(b), and a tree decomposition of $G_{\mathcal{H}(I)}$ in Figure 1(c). It is easy to verify that the given tree decomposition is optimal, and thus $tw(I) = 2$. ■

[2]In fact, this result was shown for fragments of monadic second-order logic.

A fragment $\mathcal{L}$ of first-order logic enjoys the finite treewidth model property if, for each satisfiable formula $\varphi \in \mathcal{L}$, there exists a model of φ (which can be seen as a relational instance) of finite treewidth. Let us now give our decidability result:

THEOREM 3.5. *UCQ answering under weakly-frontier-guarded DTGDs is decidable.*

PROOF. Let $\mathcal{L}_{WFG}$ be the fragment of first-order logic which can express only formulas of the form $\Phi_{D,\Sigma,Q} = (D \wedge \Sigma \wedge \neg Q)$, where D is a database, Σ is a weakly-frontier-guarded set of DTGDs, and Q is a UCQ. Since $D \cup \Sigma \models Q$ iff $\Phi_{D,\Sigma,Q}$ is unsatisfiable, it suffices to show that $\mathcal{L}_{WFG}$ enjoys the finite treewidth model property. Consider an arbitrary formula $\Phi_{D,\Sigma,Q} \in \mathcal{L}_{WFG}$. By the universality property of the disjunctive chase, $D \cup \Sigma \models Q$ iff $I \models Q$, for each $I \in \mathit{chase}(D, \Sigma)$. Hence, $\Phi_{D,\Sigma,Q}$ is satisfiable iff there exists $I \in \mathit{chase}(D, \Sigma)$ such that $(I \wedge \neg Q)$ is satisfiable. Clearly, if $\Phi_{D,\Sigma,Q}$ is satisfiable, then there exists $I \in \mathit{chase}(D, \Sigma)$ which is a model of $\Phi_{D,\Sigma,Q}$. By definition of the disjunctive chase, the instance I can be seen as the result of the (non-disjunctive) chase under D and a set of weakly-frontier-guarded (non-disjunctive) TGDs. It is implicit in [Baget et al. 2011a], where weakly-frontier-guarded (non-disjunctive) TGDs are investigated, that I has finite treewidth, and the claim follows. □

The above theorem establishes decidability of (U)CQ answering under the guarded-based classes of DTGDs that we consider in this work. However, it tells nothing about the computational complexity of our problem. Understanding the complexity of the problem under consideration will be the subject of the next sections.

4. ARBITRARY QUERIES

In this section, we focus on answering (unions of) conjunctive queries under our respective classes of DTGDs. The overview section below contains a summary of our results, and discusses results inherited from existing works. Full proofs for our novel upper and lower complexity bounds are given in Section 4.2 and 4.3, respectively.

4.1. Overview

Table I summarizes the complexity results for answering (U)CQs under the various classes of DTGDs considered in this paper. Each row corresponds to a class of DTGDs (which is decoded by substituting L for linear, ML for multi-linear, G for guarded, F for frontier, and W for weakly), while each column corresponds to a different setting of the problem. In each cell of the table, we have indicated where to find the corresponding results; UB and LB stands for upper and lower bound, respectively. Note that missing references for the upper (resp., lower) bounds are immediately inherited from the first lower-left (resp., upper-right) cell in which a reference is given. The results of this section, together with a brief description of the employed techniques, follow.

Upper Bounds:

(1) UCQ answering under weakly-frontier-guarded DTGDs is in 2EXPTIME in combined complexity (Theorem 4.1) — this is established by reduction to the unsatisfiability problem of GNFO sentences, which is 2EXPTIME-complete [Bárány et al. 2015, Theorem 3.4];
(2) UCQ answering under frontier-guarded DTGDs is in coNP in data complexity (Theorem 4.3) — this is proved by reduction to UCQ answering under GFO, which is coNP-complete in data complexity [Bárány et al. 2014, Theorem 5.1]; and
(3) UCQ answering under weakly-frontier-guarded DTGDs is in EXPTIME in data complexity (Theorem 4.4) — this is shown by first providing a reduction to UCQ

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
DID	2ExpTime	2ExpTime	2ExpTime LB: Thm. 4.8	coNP LB: [Calvanese et al. 2013, Thm. 4.5]
L	2ExpTime	2ExpTime	2ExpTime	coNP
ML	2ExpTime	2ExpTime	2ExpTime	coNP
G	2ExpTime	2ExpTime	2ExpTime	coNP
FG	2ExpTime	2ExpTime	2ExpTime	coNP UB: Thm. 4.3
WG	2ExpTime	2ExpTime	2ExpTime	ExpTime LB: [Calì et al. 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.1	2ExpTime	2ExpTime	ExpTime UB: Thm. 4.4

Table I: Complexity of answering (U)CQs under guarded-based DTGDs.

answering under GFO sentences, and then exploiting the fact that UCQ answering under GFO is feasible in exponential time in the size of the given sentence [Bárány et al. 2014, Theorem 5.1].

Lower Bounds:

(1) UCQ answering under DIDs is 2ExpTime-hard in combined complexity, even if we focus on predicates of arity at most two (Theorem 4.5) — this is established by reduction from the non-acceptance problem of an alternating exponential space Turing machine;
(2) UCQ answering under a fixed set of DIDs is 2ExpTime-hard, even if we consider predicates of arity at most two (Theorem 4.6) — this is shown by adapting the reduction in the proof of Theorem 4.5 in such a way that the constructed set of DTGDs does not depend on the Turing machine; and
(3) CQ answering under a fixed set of DIDs is 2ExpTime-hard, even if we consider predicates of arity at most three (Theorem 4.8) — we first show that UCQ answering under (arbitrary) DTGDs is reducible in polynomial time to CQ answering at the price of increasing the arity of the underlying schema by one (Lemma 4.7), and then we exploit Theorem 4.6.

Clearly, the 2ExpTime upper bound for weakly-frontier-guarded DTGDs (the most expressive formalism considered here), and the 2ExpTime lower bound in the case of a fixed set of DIDs (the weakest formalism studied in this paper), close the picture of the computational complexity of our problem w.r.t. the combined complexity, the case of bounded arity, and the case of a fixed set of DTGDs. Existing results provide us with two optimal lower bounds for the data complexity of our problem, which complete the entire picture of the complexity of (U)CQ answering.

Inherited Results:

(1) CQ answering under DIDs is coNP-hard in data complexity [Calvanese et al. 2013, Theorem 4.5][3] — in fact, this is shown for TBoxes with a single description logic axiom of the form $A_1 \sqsubseteq A_2 \sqcup A_3$, where each A_i is an atomic concept, which in turn is logically equivalent to the DID $A_1(X) \rightarrow A_2(X) \vee A_3(X)$; and
(2) CQ answering under weakly-guarded DTGDs is EXPTIME-hard in data complexity [Calì et al. 2013, Theorem 4.1] — actually, this is shown for weakly-guarded TGDs (without disjunctive heads).

From the results of this section, we observe that for the expressive formalisms under consideration, namely weakly-(frontier-)guarded DTGDs, the addition of disjunction does not have a significant effect on the complexity of (U)CQ answering. However, for the less expressive formalisms, the impact can be enormous with, e.g., a jump from NP-completeness to 2EXPTIME-completeness in the case of a fixed set of DIDs. Let us now proceed with the formal proofs of our results.

4.2. Upper Bounds

We start this section by showing the following result:

THEOREM 4.1. *UCQ answering under weakly-frontier-guarded DTGDs is in 2*EXPTIME *in combined complexity.*

PROOF (SKETCH). We provide a polynomial-time reduction to the problem of deciding whether a GNFO sentence is unsatisfiable, which in turn is in 2EXPTIME [Bárány et al. 2015]. Consider a database D, a set Σ of weakly-frontier-guarded DTGDs, and a UCQ Q. We construct a database D', a set Σ' of frontier-guarded DTGDs, and a UCQ Q', such that $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models Q'$; for the construction see Appendix B. Now, observe that a frontier-guarded DTGD σ of the form $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y}))$ can be equivalently rewritten as the sentence $\Phi_\sigma = \neg(\exists \mathbf{X}(\varphi(\mathbf{X}) \wedge \neg \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})))$, which falls in GNFO since all the free variables of $\exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})$ appear in the frontier-guard of $\varphi(\mathbf{X})$. Moreover, given a (Boolean) CQ q, $\neg q$ trivially falls in GNFO since in q there are no free variables. From the above discussion, we conclude that the sentence $\Psi_{D',\Sigma',Q'} = (D' \wedge \bigwedge_{\sigma \in \Sigma'} \Phi_\sigma \wedge \bigwedge_{q \in Q'} \neg q)$ falls in GNFO. The claim follows since $D' \cup \Sigma' \models Q'$ iff $\Psi_{D',\Sigma',Q'}$ is unsatisfiable. □

Notice that an alternative way to obtain the above result is to reduce our problem to UCQ answering under GFO sentences, which is also in 2EXPTIME [Bárány et al. 2014]. However, the translation from frontier-guarded DTGDs to GNFO is straightforward, whereas the translation to GFO is more involved, as GFO imposes tighter syntactic restrictions on formulas.

We now focus on the data complexity of UCQ answering under (weakly-)frontier-guarded DTGDs. It is known that CQ answering under frontier-guarded TGDs can be reduced in linear time to UCQ answering under GFO sentences [Baget et al. 2011b]. Interestingly, the same reduction (with minor adaptations), that can be found in Appendix B, can be employed even if we consider frontier-guarded DTGDs:

LEMMA 4.2. *UCQ answering under frontier-guarded DTGDs can be reduced in linear time to UCQ answering under GFO sentences.*

By exploiting the above auxiliary result, we are now ready to establish the desired upper bounds for the data complexity of our problem.

[3]The proof of this result is actually an adaptation of the proof for the coNP-hardness of instance checking for $\mathcal{ALE}$ presented in [Donini et al. 1994], which in turn relies on a construction given in [Schaerf 1993].

THEOREM 4.3. *UCQ answering under frontier-guarded DTGDs is in co*NP *in data complexity.*

PROOF. The result follows from Lemma 4.2, and the fact that UCQ answering under GFO sentences is in coNP in data complexity [Bárány et al. 2014]. □

We now focus on weakly-frontier-guarded DTGDs:

THEOREM 4.4. *UCQ answering under weakly-frontier-guarded DTGDs is in* EXPTIME *in data complexity.*

PROOF. Consider a database D, a set Σ of weakly-frontier-guarded DTGDs, and a UCQ Q. We first reduce our problem to UCQ answering under frontier-guarded DTGDs by replacing the non-affected variables in the DTGDs of Σ with all possible constants occurring in D. In other words, we partially ground the set Σ, and we obtain a set Σ' of frontier-guarded DTGDs. Clearly, Σ' is of exponential size in the number of non-affected variables, but of polynomial size in $|dom(D)|$. By Lemma 4.2, there exists a linear translation τ such that $D \cup \Sigma' \models Q$ iff $D \cup \tau(\Sigma') \models \tau(Q)$, where $\tau(\Sigma')$ is a GFO sentence and $\tau(Q)$ a UCQ. It is important to say that, although $|\tau(Q)|$ depends on D, the size of each CQ in $\tau(Q)$ does not depend on D. As shown in [Bárány et al. 2014], UCQ answering under GFO sentences depends doubly-exponentially on the size of each CQ of the given UCQ and the maximum arity of the schema, and exponentially on the size of the given GFO sentence. Since the size of each query of $\tau(Q)$ and the maximum arity of the schema are constant in D, while the size of $\tau(\Sigma')$ is polynomial in D, we get an EXPTIME upper bound w.r.t. D, and the claim follows. □

Recall that UCQ answering under DIDs is coNP-hard in data complexity [Calvanese et al. 2013], while for weakly-guarded DTGDs is EXPTIME-hard [Calì et al. 2013]. Thus, the picture of the data complexity is now complete.

4.3. Lower Bounds

We now present a series of strong 2EXPTIME lower bounds for answering arbitrary (U)CQs under DIDs. The general idea is to force a (binary) tree structure to appear in every model of the disjunctive chase, representing a sequence of configurations. Each such configuration is in turn represented by a full binary tree of depth n, such that the 2^n leaves represent the tape contents, as well as the cursor position and state, of the simulated Turing machine. We will then construct a UCQ, where each disjunct checks if there is an error in the tree structure (i.e., if it does not represent a valid computation of the Turing machine).

THEOREM 4.5. *UCQ answering under DIDs is 2*EXPTIME*-hard in combined complexity, even for predicates of arity at most two.*

PROOF. The proof is via reduction of the non-acceptance problem of an alternating exponential space Turing machine M on the empty input. Let $M = (S, \Lambda, \delta, s_0)$ be an alternating Turing machine as defined in Section 2. For simplicity, we assume that $\Lambda = \{0, 1, \sqcup\}$. We also assume that M is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly 2^n tape cells, where $n > 1$. Furthermore, we assume that a rejecting configuration does not have a subsequent configuration, while an accepting configuration has only itself as a subsequent configuration[4]. We

[4]Strictly speaking, it is not possible for a configuration to have itself as a subsequent configuration, since the cursor must move either to the left or to the right. However, we can assume that from an accepting configuration C, after two steps we always visit the same configuration C; simply move the cursor to the left

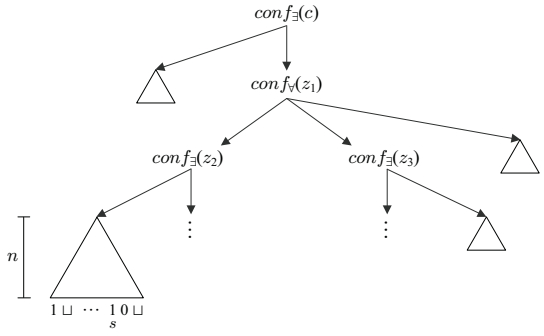

Fig. 2: Representation of the computation tree in the proof of Theorem 4.5.

also assume that $s_0 \in S_\exists$, and also that every universal configuration is followed by two existential configurations and vice versa (note that this allows us to conceive the transition relation as a function, i.e., $\delta : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$). Finally, for technical reasons, which will be clarified later, we assume that M never touches the first and the last cells of the tape, and also starts with its head at the second cell. The above assumptions can be made, without sacrificing the generality of our proof, since the non-acceptance problem of M remains 2EXPTIME-hard. Our goal is to construct a database D, a set Σ of DTGDs, and a UCQ Q such that $D \cup \Sigma \models Q$ iff M rejects, $\mathsf{N}(\Sigma)$ is a set of DIDs, and $sch(\mathsf{N}(\Sigma))$ consists of unary and binary predicates.

The general idea underlying our proof is to construct trees, which encode possible computation trees of M, by chasing D and Σ, and then check their consistency via the query Q. To each configuration node v, which represents the configuration C_v of M, we attach a configuration tree, that is, a full binary tree of depth n, and thus at its n-th level there are exactly 2^n nodes which represent the cells of the tape of M in C_v. Furthermore, for each cell we guess whether the cursor of M is at this cell, and if so, we label the cell with the state s of C_v. The above informal description is illustrated in Figure 2. We now proceed with the formal construction of D, Σ and Q.

The Database D. Let $D = \{child(a, b), child(b, c), conf_\exists(c)\}$, where $c \in \mathbf{C}$ is a special constant that represents the initial configuration. Notice that, by assumption, the initial configuration is existential. For technical reasons, which will become clear later, we need the initial configuration to have two ancestor nodes.

The Set Σ. The predicates that we are going to use are self-explanatory, and thus we proceed with the construction of Σ without describing the intuitive meaning of the predicates of $sch(\Sigma)$.

— Each configuration has two successor configurations, such that a universal configuration is followed by existential configurations, and vice-versa. Note that for existential configurations, we generate two models for the successors, whereas for universal configurations, both are forced to appear in the same model:

$$\begin{aligned} conf_\exists(X) &\rightarrow \exists Y\ child_1(X, Y), conf_\forall(Y) \vee \exists Y\ child_2(X, Y), conf_\forall(Y), \\ conf_\forall(X) &\rightarrow \exists Y \exists Z\ child_1(X, Y), conf_\exists(Y), child_2(X, Z), conf_\exists(Z), \\ child_i(X, Y) &\rightarrow child(X, Y), \text{ for each } i \in \{1, 2\}. \end{aligned}$$

and then to the right (or to the right and then to the left), without changing the state of the machine or the content of the tape.

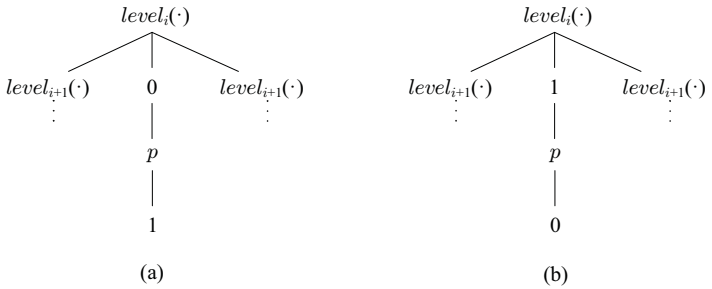

Fig. 3: Examples of navigation gadgets.

Roughly speaking, the atom $child(z_1, z_2)$ expresses the fact that z_1 is a configuration with z_2 being one of its subsequent configurations.

— Each configuration has a configuration tree, i.e., a full binary tree of depth n with 2^n leaf nodes, which simulates its tape and encodes its state:

$$conf_x(X) \rightarrow level_0(X), \text{ for each } x \in \{\exists, \forall\},$$

$$\begin{aligned}
level_i(X) &\rightarrow \exists Y_L \exists Y_R\, child_L(X, Y_L), level_{i+1}(Y_L), \\
&\qquad child_R(X, Y_R), level_{i+1}(Y_R), \text{ for each } i \in [n-1], \\
child_x(X, Y) &\rightarrow child(X, Y), \text{ for each } x \in \{L, R\}, \\
level_n(X) &\rightarrow \bigvee_{\lambda \in \Lambda} \lambda(X), \\
level_n(X) &\rightarrow cursor(X) \vee notcursor(X), \\
cursor(X) &\rightarrow \bigvee_{s \in S} s(X).
\end{aligned}$$

Notice that the last three DTGDs guess the content of the tape cells (recall that each leaf node of the configuration tree represents a cell), as well as the state of the machine and the position of the cursor.

— In order for our UCQ to be able to navigate through the constructed tree, we attach a navigation gadget to each node of a configuration tree, using the following TGDs:

$$\begin{aligned}
child_L(X, Y) &\rightarrow \exists Y_1 \exists Y_2 \exists Y_3\, child(Y, Y_1), child(Y_1, Y_2), child(Y_2, Y_3), 0(Y_1), p(Y_2), 1(Y_3), \\
child_R(X, Y) &\rightarrow \exists Y_1 \exists Y_2 \exists Y_3\, child(Y, Y_1), child(Y_1, Y_2), child(Y_2, Y_3), 1(Y_1), p(Y_2), 0(Y_3), \\
child(X, Y) &\rightarrow parentorchild(X, Y), parentorchild(Y, X).
\end{aligned}$$

The navigation gadget encodes whether a particular node is the left or right child of its parent. Each child on the left has a chain of length three attached to it, where the first node is labeled by 0, the second by p, and the third by 1. For each child on the right, the labels 0 and 1 are reversed. An example of a navigation gadget, which encodes the left (resp., right) child of a node, is depicted in Figure 3(a) (resp., 3(b)).

The construction of Σ is now complete. It is easy to verify that $\mathsf{N}(\Sigma)$ is a set of DIDs, and also that $sch(\mathsf{N}(\Sigma))$ consists of unary and binary predicates.

The UCQ Q. We now proceed with the construction of Q, which ensures that each model of $D \cup \Sigma$ encodes an invalid computation tree (and thus $D \cup \Sigma \models Q$ iff M rejects the empty input). Q consists of the following disjuncts:

(1) $Q_{initial}$ for checking that the initial configuration is invalid, i.e., that the cursor is not at the second cell or the tape is not empty or the state is not s_0;
(2) Q_{cursor} for checking that more than one cell is pointed by the cursor;
(3) Q_{trans} for checking that the transition function of M is violated; and
(4) $Q_{inertia}$ for checking that the cells not under the cursor do not keep their old value during a transition.

In the sequel, for a binary predicate r, we write $r^i(X,Y)$, where $i > 0$, as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1} \left(r(X,Z_1) \wedge \ldots \wedge r(Z_{i-1},Y)\right)$; for $i \leqslant 0$, $r^i(X,Y)$ is defined as $X = Y$. Note that our definition of CQs does not include equalities, however, we can simply remove the equality by replacing all occurrences of X with Y. Henceforth, we will interpret an atom $X = Y$ in this way. Let us now formalize Q.

(1) $Q_{initial}$ contains four disjuncts. The first checks whether the cursor is not at the second tape cell; the second checks whether the second tape cell is labeled with anything but the stating state s_0; the third checks that some tape cell contains a non-blank symbol; and the fourth checks that the cursor is at some tape cell except the second one (i.e., that there are multiple cursor positions). $Q_{initial}$ is defined as follows; recall that $c \in \mathbf{C}$ represents the initial configuration of M:

$$\begin{aligned}
&\exists X \exists Y \left(child_L^{n-1}(c,X) \wedge child_R(X,Y) \wedge nocursor(Y)\right) \vee \\
&\quad \bigvee_{s \in (S \setminus \{s_0\})} \exists X \exists Y \left(child_L^{n-1}(c,X) \wedge child_R(X,Y) \wedge s(Y)\right) \vee \\
&\quad\quad \bigvee_{\lambda \in \Lambda \setminus \{\sqcup\}} \exists X \left(child^n(c,X) \wedge \lambda(X)\right) \vee \\
&\quad\quad\quad \bigvee_{i \in ([n] \setminus \{n-1\})} \exists X \exists Y \exists Z \left(child_L^i(c,X) \wedge child_R^{min(1,n-i)}(X,Y) \wedge child^{n-i-1}(Y,Z) \wedge cursor(Z)\right)
\end{aligned}$$

(2) Q_{cursor} is defined as

$$\begin{aligned}
\bigvee_{i \in [n-1]} \exists X \exists Y \exists Y_L \exists Y_R \exists Z_L \exists Z_R \big(&child^i(X,Y) \wedge child_L(Y,Y_L) \wedge child_R(Y,Y_R) \wedge \\
&child^{n-i-1}(Y_L,Z_L) \wedge child^{n-i-1}(Y_R,Z_R) \wedge cursor(Z_L) \wedge cursor(Z_R)\big) .
\end{aligned}$$

Before giving the definition of Q_{trans} and $Q_{inertia}$, we first need to define two auxiliary subqueries:

— $siblings(X,Y,Z)$, which is true for three leaf nodes X, Y and Z iff they represent three subsequent tape cells in the same configuration; and
— $sameCell(X,Y)$, which is true for two leaf nodes X and Y iff they represent the same tape cell and they belong to successive configurations, i.e., if they belong to configurations C_X and C_Y, respectively, then C_Y is a successor configuration of C_X.

In the definition of $siblings$, we exploit the following key fact that can be easily verified. Every two neighboring leaves v_1 and v_2 in a full binary tree, where v_1 appears to the left of v_2, have a common ancestor node v such that v_1 (resp., v_2) can be reached from v by a path that leads to the left (resp., right) once, and only to the right (resp.,

left) afterwards. The subquery $siblings$ is defined as follows:

$$\begin{aligned}
&siblings(X,Y,Z)\\
\equiv\ &\exists P_1\exists P_2\exists P_3\exists P_4\,(child_L(P_1,X)\wedge child_R(P_1,Y)\wedge child_L(P_2,P_3)\wedge child_R(P_2,P_4)\wedge\\
&\qquad\qquad\bigvee_{1<i<n}\left(child_R^i(P_3,Y)\wedge child_L^i(P_4,Z)\right)\Bigg)\vee\\
&\exists P_1\exists P_2\exists P_3\exists P_4\,(child_L(P_1,Y)\wedge child_R(P_1,Z)\wedge child_L(P_2,P_3)\wedge child_R(P_2,P_4)\wedge\\
&\qquad\qquad\bigvee_{1<i<n}\left(child_R^i(P_3,X)\wedge child_L^i(P_4,Y)\right)\Bigg).
\end{aligned}$$

Note that the above query contains a mixture of disjunctions and conjunctions, and will be used as a conjunct in subsequent queries, yielding a query that is not strictly a UCQ. However, by unfolding via distribution, a UCQ of at most quadratic size can be obtained. Any use as the above should be seen as such an unfolded UCQ.

We proceed now with the definition of $sameCell$. In fact, $sameCell$ comes in two varieties: $sameCell_1(X,Y)$ and $sameCell_2(X,Y)$ depending on whether Y is in the first or second successor configuration from X. Let us for the moment assume that we can call a subquery $\Psi_i(X,Y)$, which expresses that if X and Y are nodes at level $i>0$ in successive configuration trees, then $\Psi_i(X,Y)$ is true iff X and Y are both either left or right children of their parent nodes. Then, for each $j\in[2]$,

$$\begin{aligned}
&sameCell_j(X,Y)\\
\equiv\ &\exists X_0\ldots\exists X_{n-1}\exists Y_0\ldots\exists Y_{n-1}\\
&\quad(level_0(X_0)\wedge level_0(Y_0)\wedge child_j(X_0,Y_0)\wedge\\
&\qquad\bigwedge_{1\leqslant i\leqslant n-1}(child(X_{i-1},X_i)\wedge child(Y_{i-1},Y_i)\wedge\Psi_i(X_i,Y_i))\wedge\\
&\qquad child(X_{n-1},X)\wedge child(Y_{n-1},Y)\wedge\Psi_i(X,Y))\,.
\end{aligned}$$

We now define the subquery $\Psi_i(X,Y)$. It verifies, for two nodes X and Y at level i in subsequent configuration trees, whether X and Y are both left children or both right children of their parents. Notice that left and right children can be distinguished by the position of their 1-labeled navigation gadget node relative to their p-labeled navigation gadget node. In particular, a node v at level i of a configuration tree and a node u at level i of a successor configuration tree are both left or both right children, if the nodes v_1 and u_1 with label 1 in their respective gadgets have a common ancestor that has distance $i{+}3$ from v_1 and $i{+}4$ from u_1. Note that for the first and second configuration in our tree, this common ancestor actually lies before the starting configuration marked by the constant c; hence, the two ancestor nodes a and b introduced in the database D. The definition of $\Psi_i(X,Y)$ follows, where variable U denotes the common ancestor:

$$\begin{aligned}
&\Psi_i(X,Y)\\
\equiv\ &\exists P_X\exists P_Y\exists T_X\exists T_Y\exists U\,\big(p(P_X)\wedge p(P_Y)\wedge 1(T_X)\wedge 1(T_Y)\wedge child^2(X,P_X)\wedge child^2(Y,P_Y)\\
&\quad parentorchild(P_X,T_X)\wedge parentorchild(P_Y,T_Y)\wedge child^{i+3}(U,T_X)\wedge child^{i+4}(U,T_Y)\big)\,.
\end{aligned}$$

We are now ready to define Q_{trans} and $Q_{inertia}$.

(3) For the definition of Q_{trans}, it is more convenient to consider the transitions of δ as *rewriting assertions* of the form

$$(x,y,z)\ \rightarrow\ ((x_1,y_1,z_1),(x_2,y_2,z_2)),$$

where $x, z \in \Lambda$, $y \in (S \times \Lambda)$, $x_i, z_i \in (\Lambda \cup (S \times \Lambda))$ and $y_i \in \Lambda$, for each $i \in [2]$. Moreover, for each $i \in [2]$, $x_i \in \Lambda$ implies $z_i \in (S \times \Lambda)$ and $x_i \in (S \times \Lambda)$ implies $z_i \in \Lambda$. Intuitively, y represents the state of the machine and the symbol read by the cursor, while x and z are the symbols to the left and right of the cursor position. In a subsequent configuration such a partial configuration (xyz) then transforms into $(x_1y_1z_1)$ and $(x_2y_2z_2)$, according to δ. Before defining Q_{trans}, we need to introduce some auxiliary notions. First, we define the set T_δ of all possible rewriting assertions that are consistent with δ. Formally, $T_\delta = \bigcup_{\tau \in \delta} f(\tau)$, where, assuming that $\tau = (s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$,

$$f(\tau) = \begin{cases} \bigcup_{x,y \in \Lambda} \{(x,(s,a),y) \to ((x,a_1,(s_1,y)),(x,a_2,(s_2,y)))\}, & \text{if } d_1 = +1, d_2 = +1, \\ \bigcup_{x,y \in \Lambda} \{(x,(s,a),y) \to ((x,a_1,(s_1,y)),((s_2,x),a_2,y))\}, & \text{if } d_1 = +1, d_2 = -1, \\ \bigcup_{x,y \in \Lambda} \{(x,(s,a),y) \to (((s_1,x),a_1,y),(x,a_2,(s_2,y)))\}, & \text{if } d_1 = -1, d_2 = +1, \\ \bigcup_{x,y \in \Lambda} \{(x,(s,a),y) \to (((s_1,x),a_1,y),((s_2,x),a_2,y))\}, & \text{if } d_1 = -1, d_2 = -1. \end{cases}$$

Let U be the set of all possible assertions (not necessarily valid rewriting assertions) of the form $(x, y, z) \to ((x_1, y_1, z_1), (x_2, y_2, z_2))$, where $x, z \in \Lambda$, $y \in (S \times \Lambda)$ and $x_i, y_i, z_i \in \Lambda \cup (S \times \Lambda)$, for each $i \in [2]$. At this point, one maybe tempted to think that the set $U \setminus T_\delta$ collects all the erroneous assertions of U that must be encoded in the query Q_{trans}; recall that the purpose of Q_{trans} is to check that the transition function of M is violated. However, $U \setminus T_\delta$ may contain an assertion of the form $\alpha = (x, (s, a), z) \to ((x_1, y_1, z_1), (x_2, y_2, z_2))$, where s is an existential state, and in T_δ there exists a rewriting assertion of the form $(x, y, z) \to ((x_1, y_1, z_1), \cdot)$ or $(x, y, z) \to (\cdot, (x_2, y_2, z_2))$. According to the definition of an accepting computation tree of M, and in particular, the way existential configurations and interpreted, it is clear that α must be conceived as a valid rewriting assertion that is consistent with δ, and thus excluded from $U \setminus T_\delta$. After eliminating all those (valid) assertions from $U \setminus T_\delta$, we obtain the set $\bar{T}_\delta$. Having $\bar{T}_\delta$ in place, we are now ready to define Q_{trans}. In the definition of Q_{trans}, we exploit the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$, if $s \in S_\exists$, and as $\wedge$, if $s \in S_\forall$. We also make use of the shorthand $is_x(X) \equiv a(X)$, if $x = a \in \Lambda$, and $is_x(X) \equiv a(X) \wedge s(X)$, if $x = (s, a) \in (S \times \Lambda)$. Q_{trans} is defined as follows:

$$\bigvee_{(x,y=(s,a),z) \to ((x_1,y_1,z_1),(x_2,y_2,z_2)) \in \bar{T}_\delta} \; \overset{s}{\underset{i \in \{1,2\}}{\odot}} \; \exists X \exists Y \exists X_L \exists X_R \exists Y_L \exists Y_R$$
$$(\mathit{sameCell}_i(X, Y) \wedge \mathit{siblings}(X_L, X, X_R) \wedge \mathit{siblings}(Y_L, Y, Y_R) \wedge$$
$$\mathit{is}_x(X_L) \wedge \mathit{is}_y(X) \wedge \mathit{is}_z(X_R) \wedge \mathit{is}_{x_i}(Y_L) \wedge \mathit{is}_{y_i}(Y) \wedge \mathit{is}_{z_i}(Y_R)) \,.$$

(4) Finally, $Q_{inertia}$ is defined as

$$\bigvee_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \; \bigvee_{i \in \{1,2\}} \; \exists X \exists Y \, (\mathit{sameCell}_i(X, Y) \wedge \mathit{notcursor}(X) \wedge a(X) \wedge a'(Y)) \,.$$

Notice that, by employing the above query, we do not guarantee the inertia of the first and the last cells of the tape. This is not a problem since, by assumption, M never touches those cells, and also starts with its cursor at the second cell.

The definition of Q is now complete. It is easy to verify that Q is a UCQ.

By construction, and also the assumption that a rejecting configuration does not have a subsequent configuration, but an accepting one has only itself as a subsequent configuration, we get that $D \cup \Sigma \models Q$ iff M rejects. More precisely, if M accepts, then there exists $I \in \mathit{chase}(D, \Sigma)$ that encodes an accepting computation tree of M.

Thus, $I \not\models Q$, which implies that $D \cup \Sigma \not\models Q$. Conversely, if M rejects, then, for each $I \in chase(D, \Sigma)$, $I \models Q$. Fix an arbitrary instance $I \in chase(D, \Sigma)$. If I does not encode a computation tree of M, then, by construction, $I \models Q$. Now, towards a contradiction, assume that M rejects and I encodes an accepting computation tree of M. It is clear that $I \not\models (Q_{initial} \vee Q_{trans} \vee Q_{inertia})$. By construction, I encodes an infinite tree of configurations. But since a rejecting configuration does not have a subsequent configuration (by assumption), necessarily at least one disjunct of Q_{trans} will be entailed by I which contradicts $I \not\models Q_{trans}$. Therefore, $D \cup \Sigma \models Q$, and the claim follows. □

Notice that the constructed set Σ of DIDs in the above proof depends on the Turing machine M that we encode. In fact, the part of Σ that depends on M are the DIDs that construct the configuration trees — observe that we use n level-predicates, one for each level of the tree, and $|S|$ state-predicates, one for each state of M. Interestingly, a new construction can be devised in such a way that Σ is independent from M, and thus it is fixed. In order to avoid using n level-predicates, we can employ a single unary predicate, called $ctnode$, which stores all the nodes of a configuration tree. In order to avoid using $|S|$ state-predicates, we can replace each state by a chain of length at most $|S|$; notice that such chains can easily be constructed via two simple DIDs. Then, we can access a state by exploiting those chains, without explicitly encoding them inside a predicate (as we did in the proof of Theorem 4.5). After implementing this idea, the set of DIDs no longer depends on the Turing machine, and thus it is fixed as desired. However, we need to guarantee that each chain encodes a valid state, which can be checked by adding another disjunct in the UCQ Q; for the formal proof see Appendix B.

THEOREM 4.6. *UCQ answering under fixed sets of DIDs is 2*EXPTIME*-hard, even for predicates of arity at most two.*

The question that comes up is whether the above hardness result can be established for CQs. Interestingly, this question is answered positively, at the expense of increasing the arity of the underlying schema by one. This is done by first showing that there exists a polynomial time reduction of UCQ answering under DTGDs (not only DIDs) to CQ answering under DTGDs, which increases the arity of the underlying schema by one. Before giving the formal result, let us briefly explain how such a reduction works; the formal construction can be found in Appendix B. Each predicate p of the underlying schema is replaced by a new predicate p' of arity $arity(p) + 1$. This extra position holds a marker, either t (for *true*) or f (for *false*). Every database atom gets the value t at this new position, which implies that it is a valid atom. Moreover, each DTGD is extended so that it simply propagates this position unaltered to the head. A copy of each disjunct (i.e., CQ) of the given UCQ is added to the database (variables are replaced by new distinct constants) with f at the new position. Clearly, every CQ in the given UCQ trivially maps to the database, with f at the new position. However, all the valid atoms, i.e., the atoms that can be derived by the chase of the original database w.r.t. the original set of DTGDs, have t at the new position. We can now replace disjunctions in the UCQ by conjunctions (and thus convert the UCQ into a CQ), and just check that, for at least one query, its new position maps to t. This can simply be done by adding to the database atoms of the form $or(\cdot, \cdot, \cdot)$ encoding the logical *or*, connecting all the subqueries of the original UCQ via such or predicates, and stating that the end result must be t. The formal result follows:

LEMMA 4.7. *Consider a database D, a set Σ of DTGDs, and a UCQ Q. A database D', a set Σ' of DTGDs, where $arity(sch(\Sigma')) = arity(sch(\Sigma)) + 1$, and a CQ q can be constructed in polynomial time such that $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models q$.*

It is important to say that the construction employed in the proof of the above lemma preserves the syntactic properties of all classes of DTGDs considered in this paper. Moreover, Σ' does not depend on the database or the query, but only on the set Σ of DTGDs. Therefore, Theorem 4.6 and Lemma 4.7 imply the desired result:

THEOREM 4.8. *CQ answering under fixed sets of DIDs is 2*EXPTIME*-hard, even for predicates of arity at most three.*

Recall that UCQ answering under weakly-frontier-guarded DTGDs is in 2EXPTIME in combined complexity (Theorem 4.1). Hence, the picture of the computational complexity of CQ answering under our guarded-based classes of DTGDs is now complete. Interestingly, Theorem 4.8 closes an open question stated in [Bárány et al. 2014], regarding the complexity of query answering under fixed GFO sentences. It was shown that the problem in question is already PSPACE-hard for CQs, and in EXPTIME in case of a restricted class of CQs. However, the exact complexity was left as an open problem. Clearly, the above result gives 2EXPTIME-completeness since query answering under GFO is in 2EXPTIME in general.

COROLLARY 4.9. *(U)CQ answering under fixed GFO sentences is 2*EXPTIME*-complete.*

5. BOUNDED TREEWIDTH QUERIES

Several subclasses of CQs have been considered in the literature, with the aim of reducing the complexity of classical database problems such as query evaluation and query containment. Such a well-known fragment is the class of CQs of *bounded treewidth (BTWCQs)*, i.e., the treewidth of their hypergraph is bounded by an integer constant; see, e.g., [Chekuri and Rajaraman 2000]. Let us recall that the hypergraph $\mathcal{H}(q)$ of a query q is defined in the same way as the hypergraph $\mathcal{H}(I)$ of an instance I (see Section 3). Our goal in this section is to understand whether the complexity of (U)CQ answering under our guarded-based classes of DTGDs is affected if we focus on queries of bounded treewidth.

Unfortunately, queries of bounded treewidth do not have the expected positive impact on the complexity of our problem. Table I, which summarizes the complexity for answering (U)CQs, holds even if we consider BTW(U)CQs. As you can observe, the data complexity for all the classes of DTGDs under consideration is obtained from existing results. More precisely, the coNP-hardness for DIDs is obtained from [Calvanese et al. 2013, Theorem 4.5], where it is shown that CQ answering under a single DID of the form $p_1(X) \rightarrow p_2(X) \vee p_3(X)$, is already coNP-hard, even if the input query is fixed (and thus of bounded treewidth). The coNP upper bound for frontier-guarded DTGDs follows from Theorem 4.3. The EXPTIME-hardness for weakly-guarded DTGDs is inherited from [Calì et al. 2013, Theorem 4.1], where it is shown that CQ answering under a fixed set of weakly-guarded TGDs is EXPTIME-hard, even if the input query is a single atom. The EXPTIME upper bound for weakly-frontier-guarded DTGDs follows from Theorem 4.4.

Although the data complexity of our problem can be settled by exploiting known results, this is not the case for the other settings of the problem. The best known upper bound is the 2EXPTIME upper bound for answering arbitrary UCQs under weakly-frontier-guarded DTGDs (Theorem 4.1). This result, combined with the fact that CQ answering under guarded TGDs is 2EXPTIME-hard in the combined complexity [Calì et al. 2013, Theorem 6.1], even for atomic queries of the form $\exists X p(X)$ (and thus of bounded treewidth), closes the combined complexity for (weakly-)(frontier-)guarded DTGDs. However, the above lower bound for guarded DTGDs is not strong enough to complete the complexity picture of our problem. Interestingly, the queries employed

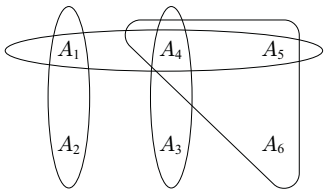

Fig. 4: The hypergraph $\mathcal{H}(q)$.

in the proofs of Theorems 4.5, 4.6 and 4.8, are already of bounded treewidth. This immediately implies that the above theorems hold also for (U)CQs of bounded treewidth, and the next result follows:

THEOREM 5.1. *BTW(U)CQ answering under fixed sets of DIDs is 2*EXPTIME*-hard, even for predicates of arity at most three (two).*

Another key class of queries is the class of CQs of *bounded hypertree-width* [Gottlob et al. 2002]. The hypertree-width is a measure of how close to acyclic a hypergraph is, analogous to treewidth for graphs. The hypertree-width of a CQ is less than or equal to its treewidth. Since all the upper bounds in Table I hold for arbitrary (U)CQs, we get that arbitrary (U)CQs, (U)CQs of bounded treewidth and (U)CQs of bounded hypertree-width are indistinguishable w.r.t. the complexity of query answering under our guarded-based classes of DTGDs.

6. ACYCLIC QUERIES

In this section, we focus on another important subclass of conjunctive queries, which has been proposed with the aim of reducing the complexity of query evaluation and query containment, namely the class of *acyclic* CQs; see, e.g., [Chekuri and Rajaraman 2000]. Our goal is to understand how the complexity of (U)CQ answering under our guarded-based classes of DTGDs is affected if we focus on acyclic queries. The acyclicity of a CQ q is defined via the acyclicity of its hypergraph $\mathcal{H}(q)$. Informally, a hypergraph is acyclic if it can be reduced to the empty hypergraph by iteratively eliminating some non-maximal hyperedge, or some vertex contained in at most one hyperedge; this procedure is known as the GYO algorithm. The formal definition is as follows.

Consider a hypergraph $\mathcal{H} = (V, H)$. A hyperedge $h \in H$ is called an *ear* if (1) $h \cap h' = \varnothing$, for each $h' \in H \setminus \{h\}$, or (2) there exists $h' \in H \setminus \{h\}$ such that $h \cap (\bigcup_{h'' \in H \setminus \{h\}} h'') \subseteq h'$. The *GYO-reduct* of $\mathcal{H}$, denoted $GYO(\mathcal{H})$, is obtained by applying exhaustively the following two steps until $\mathcal{H}$ has no ears: (1) choose an arbitrary ear h of $\mathcal{H}$; and (2) let $\mathcal{H} = (V', H \setminus \{h\})$, where $V' = \bigcup_{h' \in H \setminus \{h\}} h'$. We say that $\mathcal{H}$ is acyclic if $GYO(\mathcal{H}) = (\varnothing, \varnothing)$. A CQ q is *acyclic (ACQ)* if $\mathcal{H}(q)$ is acyclic.

Example 6.1 (*Acyclic Query*). Consider the CQ

$$q \;=\; \exists A_1 \ldots \exists A_6\, p(A_1, A_2) \wedge p(A_3, A_4) \wedge s(A_1, A_4, A_5) \wedge s(A_4, A_5, A_6).$$

The hypergraph $\mathcal{H}(q)$ is shown in Figure 4. Clearly, the hyperedge $\{A_4, A_5, A_6\}$ is an ear, and thus can be eliminated. Now, both $\{A_1, A_2\}$ and $\{A_3, A_4\}$ are ears and can be eliminated. Finally, the remaining hyperedge $\{A_1, A_4, A_5\}$ is an ear, since it does not intersect with any other hyperedge. Thus, $GYO(\mathcal{H}(q)) = (\varnothing, \varnothing)$. ∎

6.1. Overview

Table II summarizes the complexity results for answering A(U)CQs under the various DTGD formalisms considered in this paper. Compared with the results in Table I, it

	Combined Complexity	Bounded Arity	Fixed Theory	Data Complexity
DID	2ExpTime LB: Thm. 6.4	ExpTime	ExpTime LB: Thm. 6.6	coNP LB: [Calvanese et al. 2013, Thm. 4.5]
L	2ExpTime	ExpTime	ExpTime	coNP
ML	2ExpTime	ExpTime	ExpTime	coNP
G	2ExpTime	ExpTime	ExpTime	coNP
FG	2ExpTime	2ExpTime LB: Thm. 6.5	ExpTime	coNP UB: Thm. 4.3
WG	2ExpTime	ExpTime UB: Thm. 6.2	ExpTime	ExpTime LB: [Calì et al. 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.1	2ExpTime	ExpTime UB: Thm. 6.3	ExpTime UB: Thm. 4.4

Table II: Complexity of answering acyclic (U)CQs under guarded-based DTGDs. The ExpTime upper bound provided by Theorem 6.2 holds with the additional assumption that the number of body-variables is bounded.

is immediately apparent that the combined and the data complexity do not change if we restrict ourselves to acyclic queries. The complexity decreases from 2ExpTime to ExpTime for the less expressive classes of DTGDs, namely DIDs, (multi-)linear and guarded, in the case of predicates of bounded arity, and also for all the classes if we consider a fixed set of DTGDs. The novel results of this section follow:

Upper Bounds:

(1) AUCQ answering under weakly-guarded sets of DTGDs with bounded number of body-variables is in ExpTime if we focus on predicates of bounded arity[5] (Theorem 6.2) — this is shown by reduction to satisfiability of the guarded fragment of first-order logic, which is in ExpTime in case of predicates of bounded arity [Grädel 1999]; and
(2) AUCQ answering under fixed weakly-frontier-guarded sets of DTGDs is in ExpTime (Theorem 6.3) — by reduction to UCQ answering under (non-fixed) GFO sentences with predicates of bounded arity, where in the constructed (partially acyclic) query the cyclic part is fixed.

Lower Bounds:

(1) ACQ answering under DIDs is 2ExpTime-hard in combined complexity (Theorem 6.4) — this is established by simulating the behavior of an alternating exponential space Turing machine by means of a set of DIDs and an ACQ;

[5]Notice that for guarded DTGDs the number of body-variables is bounded by the arity of the underlying schema. Thus, by considering predicates of bounded arity, we implicitly bound the number of body-variables.

(2) ACQ answering under frontier-guarded DTGDs is 2EXPTIME-hard, even if we consider predicates of bounded arity (Theorem 6.5) — this is shown by reduction from CQ answering under frontier-guarded DTGDs; and
(3) ACQ answering under fixed sets of DIDs is EXPTIME-hard (Theorem 6.6) — again by simulating the behavior of an alternating linear space Turing machine.

The EXPTIME upper bound for guarded DTGDs in the case of predicates of bounded arity, and for fixed sets of weakly-frontier-guarded DTGDs, and the EXPTIME lower bound for fixed sets of DIDs, close the picture of the computational complexity of our problem for DIDs, (multi-)linear and guarded in the case of bounded arity, and for all classes in the case of a fixed set of DTGDs. The missing upper bounds are inherited from results of Section 4; for details see Table II. Finally, the following results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of A(U)CQ answering is completed.

Inherited Results:

(1) ACQ answering under DIDs is coNP-hard in data complexity [Calvanese et al. 2013, Theorem 4.5] — the CQ employed in the proof of this result is of the form $\exists X \exists Y_1 \exists Y_2 \exists Z_1 \exists Z_2 (\bigwedge_{i \in \{1,2\}} (p_i(X, Y_i) \wedge s(Y_i) \wedge r_i(X, Z_i) \wedge t(Z_i)))$, which is clearly acyclic; and
(2) ACQ answering under weakly-guarded DTGDs is EXPTIME-hard in data complexity [Calì et al. 2013, Theorem 4.1] — the CQ used in the proof of this result consists of a single atom, and thus is trivially acyclic.

From the results of this section, we observe that the acyclicity of the query does not make the query answering problem easier w.r.t. the combined and data complexity. However, in the cases of bounded arity (for the less expressive classes) and fixed sets of DTGDs, the complexity decreases from 2EXPTIME to EXPTIME. Let us now proceed with the formal proofs of our results.

6.2. Upper Bounds

We start this section by showing the following:

THEOREM 6.2. *AUCQ answering under weakly-guarded sets of DTGDs with bounded number of body-variables is in* EXPTIME *in case of predicates of bounded arity.*

PROOF (SKETCH). Consider a database D, a weakly-guarded set Σ of DTGDs with bounded number of body-variables and predicates of bounded arity, and an AUCQ Q. The proof is by reduction to satisfiability of GFO sentences (without constants). More precisely, we construct, in polynomial time, a database D', a set Σ' of DTGDs, and a query Q' such that $\Psi_{D',\Sigma',Q'} = (D' \wedge \Sigma' \wedge \neg Q')$ falls in GFO (without constants), the arity of the underlying schema is bounded, and $D \cup \Sigma \models Q$ iff $\Psi_{D',\Sigma',Q'}$ is unsatisfiable. Since the problem of deciding whether a GFO sentence (without constants) is satisfiable is in EXPTIME in case of predicates of bounded arity [Grädel 1999], the claim follows. The formal proof can be found in Appendix C. □

Although the above result does not hold for weakly-frontier-guarded DTGDs, the next theorem shows that we get an EXPTIME upper bound if we focus on fixed weakly-frontier-guarded sets of DTGDs.

THEOREM 6.3. *AUCQ answering under fixed weakly-frontier-guarded sets of DTGDs is in* EXPTIME.

PROOF. Consider a database D, a fixed weakly-frontier-guarded set of DTGDs, and an AUCQ Q. First, we reduce our problem to UCQ answering under GFO sentences in the usual way: (1) partially ground the DTGDs of Σ in such a way that the non-affected variables are replaced by all possible combinations of constants of $dom(D)$, yielding an equivalent set of frontier-guarded DTGDs Σ'; and (2) in linear time reduce the problem of deciding whether $D \cup \Sigma' \models Q$ to the problem of deciding whether the GFO formula $(D \wedge \Psi^1_{\Sigma'})$ entails the UCQ $(Q \vee \Psi^2_{\Sigma'})$ by employing Lemma 4.2. Since Σ is fixed, we immediately get that Σ' is of polynomial size w.r.t. $|dom(D)|$. Thus, by construction, $(D \wedge \Psi^1_{\Sigma'})$ and $(Q \vee \Psi^2_{\Sigma'})$ are of polynomial size, and they use only predicates of bounded arity. Before we proceed further, we need to recall some details regarding the problem of UCQ answering under GFO as presented in [Bárány et al. 2014]. Given a GFO formula φ and a UCQ Q', the way that the problem of deciding whether $\varphi \models Q'$ is tackled is as follows: (1) Q' is rewritten as a GFO formula $\chi(Q')$ (this procedure is called *treeification*) such that $\varphi \models Q'$ iff $\varphi \models \chi(Q')$; and (2) it is checked whether the GFO formula $(\varphi \wedge \neg\chi(Q'))$ is unsatisfiable. The latter check is feasible in time $2^{\mathcal{O}((n+|\varphi|+|\chi(Q')|)\cdot m^m)}$, where $|\cdot|$ denotes the size of a formula, n is the number of predicates occurring in $(\varphi \wedge \neg\chi(Q'))$, and m is the maximum arity over all these predicates. Let us now explain how the desired upper bound is obtained. Clearly, it suffices to check whether the formula $(D \wedge \Psi^1_{\Sigma'} \wedge \neg\chi(Q) \wedge \neg\chi(\Psi^2_{\Sigma'}))$ is unsatisfiable. Recall that $(D \wedge \Psi^1_{\Sigma'})$ is of polynomial size, and also the maximum arity of the underlying schema is bounded. It remains to show that $|\chi(Q)|$ and $|\chi(\Psi^2_{\Sigma'})|$ are of polynomial size. Since Q is acyclic, we know that the treeification of Q is feasible in polynomial time [Bárány et al. 2014], and thus $|\chi(Q)|$ is polynomial. By construction, since the set Σ of DTGDs is fixed, we get that $|\Psi^2_{\Sigma'}|$ is constant. Hence, by definition of the treeification procedure, $|\chi(\Psi^2_{\Sigma'})|$ is constant, and the claim follows. □

This concludes the upper bound section for acyclic queries.

6.3. Lower Bounds

We start this section by showing that ACQ answering under DIDs is 2EXPTIME-hard. The construction is rather tedious, and hinges on the fact that, using a polynomial number of DIDs, it is possible to generate and store an exponential number of integers in binary form that can be used to index the exponentially many tape cells of an alternating EXPSPACE Turing machine. Then, the computation of such a machine can be simulated by means of a set of DIDs and a CQ, and the 2EXPTIME-hardness result follows from the fact that alternating EXPSPACE coincides with 2EXPTIME.

THEOREM 6.4. *ACQ answering under DIDs is 2*EXPTIME*-hard in combined complexity.*

PROOF. The proof is via reduction of the non-acceptance problem of an alternating exponential space Turing machine M on the empty input. Let $M = (S, \Lambda, \delta, s_0)$ be an alternating Turing machine as defined in Section 2. For technical clarity, we adopt the assumptions made in the proof of Theorem 4.5, with the difference that M may touch the first and the last cells of the tape, and may also start with its head at the second cell. Our goal is to construct a database D, a set Σ of DTGDs, and an AUCQ Q such that $D \cup \Sigma \models Q$ iff M rejects, and $\mathsf{N}(\Sigma)$ is a set of DIDs. By exploiting the construction in the proof of Lemma 4.7, a database D', a set Σ', and an ACQ q can be constructed in polynomial time such that $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models q$, and $\mathsf{N}(\Sigma')$ is a set of DIDs, which proves the claim. We proceed with the construction of D, Σ and Q.

The general idea of the proof is to construct trees, which encode possible computation trees of M, by chasing D with Σ, and then use the query Q to check the consistency

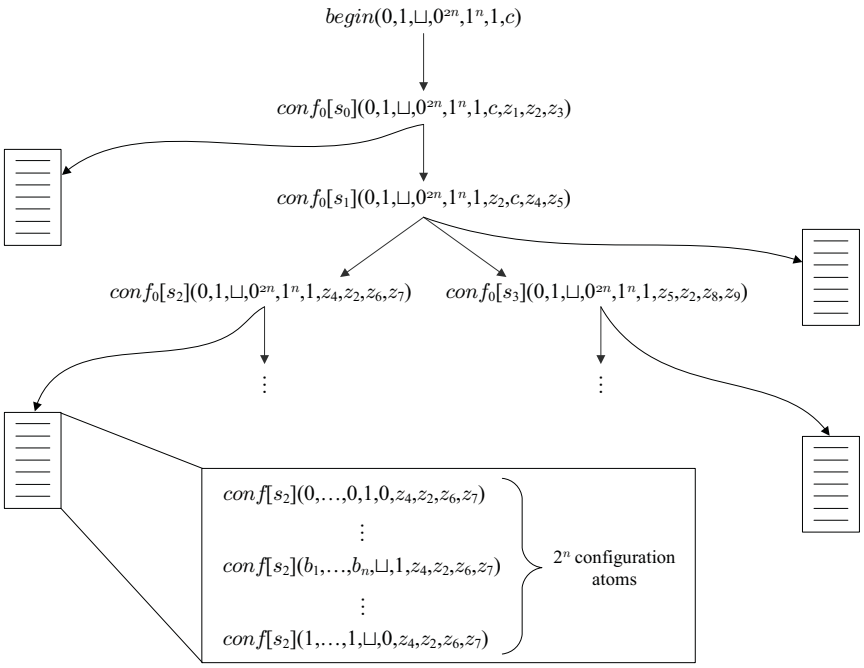

Fig. 5: Computation tree.

of those trees. In order to represent configurations of M, we will use atoms of the form $conf[s](b_1, \ldots, b_n, a, h, t, p, n_1, n_2)$, where $s \in S$ is the state of the encoded configuration and is part of the predicate, $(b_1, \ldots, b_n) \in \{0,1\}^n$ is an integer of $\{0, \ldots, 2^n - 1\}$ in binary encoding that represents the index of the encoded cell with a being its content, $h \in \{0,1\}$ and $h = 1$ iff the cursor of M is at the encoded cell, and t, p, n_1 and n_2 represent the current (t for this), the previous and the next two configurations, respectively. For example, assuming that $n = 3$, $conf[s](1,0,1,\sqcup,1,z_2,z_1,z_3,z_4)$, where $\{z_1, \ldots, z_4\} \subset \mathbf{N}$, says that the state of the configuration z_2 is s, the fifth cell contains the blank symbol, the cursor is at the fifth cell, the previous configuration of z_2 is z_1, and the next two configurations of z_2 are z_3 and z_4.

The Database D. Let the database D contain a single atom $begin(0, 1, \sqcup, \mathbf{0}^{2n}, \mathbf{1}^n, 1, c)$; for a constant x, $\mathbf{x}^k$ denotes the sequence $x, \ldots, x$ with k occurrences of x. The first three constants 0, 1 and $\sqcup$ will allow us to have access to the symbols of Λ without explicitly mentioning them in the DTGDs; recall that DIDs are constant-free. The $2n$ zeros and n ones are auxiliary constants that will allow us to generate, for each configuration of M, the 2^n integers in binary that will be used to index the tape cells that M can touch. The constant 1 will be used to ensure that exactly one tape cell is pointed by the cursor. Finally, c is a constant that represents the initial configuration.

The Set Σ. We proceed now with the construction of Σ. In the sequel, for brevity, we write $\mathbf{X}^k$ for the sequence of variables $X_1, \ldots, X_k$.

— We first construct the initial configuration with the TGD

$$begin(Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, T) \rightarrow \exists P \exists N_1 \exists N_2 \; conf_0[s_0](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, T, P, N_1, N_2).$$

Notice that in the head of the above TGD we use the auxiliary predicate $conf_0[s]$, where $s \in S$, and not the predicate $conf[s]$. This is because we first want to generate all the possible trees that may encode a computation tree of M, where the nodes are labeled with auxiliary atoms of the form $conf_0[s](0, 1, \sqcup, \mathbf{0}^{2n}, \mathbf{1}^n, 1, z_2, z_1, z_3, z_4)$. Such an atom, which is associated with the configuration z_2, contains all the auxiliary constants that will allow us to generate, via an additional set of DTGDs, all the 2^n atoms of the form $conf[s](b_1, \ldots, b_n, a, h, z_2, z_1, z_3, z_4)$ that perfectly describe the configuration z_2. The above informal description is illustrated in Figure 5.

— The trees labeled by atoms of the form $conf_0[s](\ldots)$ are generated by the following DTGDs: for each $s \in S_\forall$,

$$conf_0[s](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, T, P, N_1, N_2) \rightarrow \bigvee_{(s_1,s_2)\in S_\exists \times S_\exists} \exists N_3 \exists N_4 \exists N_5 \exists N_6\ conf_0[s_1](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, N_1, T, N_3, N_4), conf_0[s_2](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, N_2, T, N_5, N_6),$$

and for each $s \in S_\exists$,

$$conf_0[s](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, T, P, N_1, N_2) \rightarrow \bigvee_{s'\in S_\forall} \exists N_3 \exists N_4\ conf_0[s'](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, N_1, T, N_3, N_4) \vee \exists N_3 \exists N_4\ conf_0[s'](Z, O, B, \mathbf{Z}^{2n}, \mathbf{O}^n, H, N_2, T, N_3, N_4).$$

— The 2^n atoms, which perfectly describe a certain configuration, are generated as follows. First, we employ a stratified set of DIDs, consisting of n levels, in order to generate all the necessary tuples, and store them in predicates of the form $conf_n[s]$. In particular, for each $i \in \{0, \ldots, n-1\}$, and for each $s \in S$:

$$conf_i[s](Z, O, B, \mathbf{Z}^{2n-2i}, \mathbf{O}^{n-i}, \mathbf{X}^i, H, T, P, N_1, N_2) \rightarrow \varphi_1 \vee \varphi_2,$$

where

$$\varphi_1 = conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, Z_{2n-2i-1}, H, T, P, N_1, N_2), conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, O_{n-i}, Z_{2n-2i-1}, T, P, N_1, N_2)$$

and

$$\varphi_2 = conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, Z_{2n-2i-1}, Z_{2n-2i}, T, P, N_1, N_2), conf_{i+1}[s](Z, O, B, \mathbf{Z}^{2n-2i-2}, \mathbf{O}^{n-i-1}, \mathbf{X}^i, O_{n-i}, H, T, P, N_1, N_2).$$

Now, having the relation $conf_n[s]$, for each $s \in S$, in place, we are ready to generate the completed $conf[s]$ atoms by guessing the cell content of each cell. This can be done via the following DTGDs: for each $s \in S$,

$$conf_n[s](Z, O, B, \mathbf{X}^n, H, T, P, N_1, N_2) \rightarrow \bigvee_{a\in\Lambda} conf[s](\mathbf{X}^n, f(a), H, T, P, N_1, N_2),$$

where $f(0) = Z$, $f(1) = O$ and $f(\sqcup) = B$.

The construction of Σ is now complete. It is easy to verify that $\mathsf{N}(\Sigma)$ is a set of DIDs. This is because, for each $\sigma \in \Sigma$, σ is constant-free, $|body(\sigma)| = 1$, and there are no repeated variables in the atoms occurring in σ.

The AUCQ Q. It remains to check, via the AUCQ Q, that each model of $D \cup \Sigma$ encodes a tree that represents an invalid computation tree of M. Roughly speaking, Q consists of the following disjuncts:

(1) $Q_{initial}$ for checking that in the initial configuration the cursor is not at the first cell, and the tape is not empty;
(2) Q_{trans} for checking that the transition function of M is violated;
(3) Q_{move} for detecting a wrong move of the cursor during a transition, i.e., a move which is neither to the left nor to the right; and
(4) $Q_{inertia}$ for checking that the tape cells not under the cursor do not keep their old values during a transition.

Let us now formalize the components of Q.

(1) $Q_{initial}$ is defined as (recall that $c \in \mathbf{C}$ represents the initial configuration)

$$\exists C \exists P \exists N_1 \exists N_2 \, (conf[s_0](\mathbf{0}^n, C, 0, c, P, N_1, N_2)) \vee \bigvee_{a \in \Lambda \setminus \{\sqcup\}} \exists \mathbf{X}^n \exists H \exists P \exists N_1 \exists N_2 \, (conf[s_0](\mathbf{X}^n, a, H, c, P, N_1, N_2)) \, .$$

Notice that the first disjunct encodes the fact the the cursor is not at the first cell, and this is sufficient since, by construction, exactly one cell is pointed by the cursor.

(2) With Q_{trans} we need to check that during a transition of M the state and the cell content has changed appropriately, and the head has moved to the correct position. Note that two subsequent cell positions agree on some initial portion of their binary encoding, which for the preceding cell is then followed by a 0 and then all 1s, and the other way around for the succeeding cell. We can thus connect the cell under the head in one configuration and the same cell in the next configuration, plus the appropriate adjacent cell in the next configuration. In the definition of Q_{trans}, we make use of the functions $f, g : \{-1, +1\} \to \{0, 1\}$, where $f(-1) = g(+1) = 1$ and $f(+1) = g(-1) = 0$. Also, we make use of the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$, if $s \in S_\exists$, and as $\wedge$, if $s \in S_\forall$. We also need to define the complement of δ relative to universal and existential states, denoted $\bar{\delta}_\forall$ and $\bar{\delta}_\exists$, respectively. Formally, $\bar{\delta}_\forall : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_\forall$, not occurring in δ. Analogously, $\bar{\delta}_\exists : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ consists of all the transition rules of the form $(s, a) \to ((s_1, a_1, d_1), (s_2, a_2, d_2))$, where $s \in S_\exists$, for which there is no transition rule of the form $(s, a) \to ((s_1, a_1, d_1), \cdot)$ or $(s, a) \to (\cdot, (s_2, a_2, d_2))$ in δ. Q_{trans} is defined as

$$\bigvee_{(s,a) \to ((s_1,a_1,d_1),(s_2,a_2,d_2)) \in (\bar{\delta}_\forall \cup \bar{\delta}_\exists)} \; \overset{s}{\bigodot}_{i \in \{1,2\}} \; \bigvee_{0 < j \leqslant n} \exists \mathbf{X}^{n-j} \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2' \exists N_1'' \exists N_2'' \exists C$$
$$\big(conf[s](\mathbf{X}^{n-j}, f(d_i), g(d_i)^{j-1}, a, 1, T, P, N_1, N_2) \wedge$$
$$conf[s_i](\mathbf{X}^{n-j}, f(d_i), g(d_i)^{j-1}, a_i, 0, N_i, T, N_1', N_2') \wedge$$
$$conf[s_i](\mathbf{X}^{n-j}, g(d_i), f(d_i)^{j-1}, C, 1, N_i, T, N_1'', N_2'') \big) \, .$$

(3) The idea underlying Q_{move}, is to check that in a certain configuration the cursor points the k-th cell, while in the previous configuration the cursor is neither at the

$(k+1)$-th cell, nor at the $(k-1)$-th cell. The formal definition follows:

$$\bigvee_{0<i,j\leqslant n} \bigvee_{(s,s')\in S\times S} \exists \mathbf{X}^{n-i} \exists C \exists T \exists P \exists N_1 \exists N_2 \exists C' \exists P' \exists N_1' \exists N_2'$$
$$\exists \mathbf{Y}^{n-j} \exists C'' \exists P'' \exists N_1'' \exists N_2'' \exists C''' \exists P''' \exists N_1''' \exists N_2'''$$
$$\big(conf[s](\mathbf{X}^{n-i}, 0, \mathbf{1}^{i-1}, C, 1, T, P, N_1, N_2) \wedge$$
$$conf[s'](\mathbf{X}^{n-i}, 1, \mathbf{0}^{i-1}, C', 0, P, P', N_1', N_2') \wedge$$
$$conf[s](\mathbf{Y}^{n-j}, 1, \mathbf{0}^{j-1}, C'', 1, T, P'', N_1'', N_2'') \wedge$$
$$conf[s'](\mathbf{Y}^{n-j}, 0, \mathbf{1}^{i-1}, C''', 0, P, P'', N_1''', N_2''')\big) .$$

Intuitively, the first and the third atom of a disjunct in the above query access the cell k, while the second and the fourth atom access the cell $(k+1)$ and $(k-1)$, respectively. Notice that, although we do not force $(\mathbf{X}^{n-i}, 0, \mathbf{1}^{i-1})$ and $(\mathbf{Y}^{n-j}, 1, \mathbf{0}^{j-1})$ to have the same image during the evaluation of the query (otherwise, the above query will be cyclic), this is guaranteed by the fact that only one cell is pointed by the cursor (this holds by construction).

(4) Finally, $Q_{inertia}$ is defined as

$$\bigvee_{(s,s')\in S\times S} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in\{1,2\}} \exists \mathbf{X}^n \exists H \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2'$$
$$(conf[s](\mathbf{X}^n, a, 0, T, P, N_1, N_2) \wedge conf[s'](\mathbf{X}^n, a', H, N_i, T, N_1', N_2'))$$

This completes the construction of Q. It is not difficult to verify that every disjunct of Q is an ACQ, and thus Q is acyclic.

By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that $D \cup \Sigma \models Q$ iff M rejects I (see the argument given in the proof of Theorem 4.5, which shows that the employed construction is correct). □

We now focus on frontier-guarded DTGDs, and show that query answering remains 2ExpTime-hard, even if we consider acyclic queries and predicates of bounded arity.

THEOREM 6.5. *ACQ answering under frontier-guarded DTGDs is 2*ExpTime*-hard, even for predicates of bounded arity.*

PROOF. The claim can be easily established by exploiting the fact that a CQ can be conceived as a frontier-guarded TGD, and give a reduction from CQ answering under frontier-guarded DTGDs, which is 2ExpTime-hard, even for predicates of bounded arity. Consider a database D, a set Σ of DTGDs, where the arity of the predicates of $sch(\Sigma)$ is bounded by an integer constant, and a CQ $q = \exists \mathbf{X}\, \varphi(\mathbf{X})$. It is straightforward to see that $D \cup \Sigma \models q$ iff $D \cup \Sigma \cup \{\sigma_q\} \models q'$, where σ_q is the frontier-guarded TGD $\varphi(\mathbf{X}) \rightarrow p$, with p being 0-ary predicate not occurring in $sch(\Sigma)$, and $q' = p$. Since q' is trivially acyclic, the claim follows. □

The last lower bound for A(U)CQ answering is the ExpTime-hardness when we focus on fixed sets of DIDs. This is done by simulating the behavior of an alternating linear space Turing machine. The desired result follows since alternating linear space already coincides with ExpTime. For the formal proof see Appendix C.

THEOREM 6.6. *ACQ (resp., AUCQ) answering under fixed sets of DIDs is* ExpTime*-hard, even for predicates of arity at most three (resp., two).*

	Combined Complexity	Bounded Arity	Data Complexity
DID	ExpTime LB: Thm. 7.14	PTime LB: Thm. 7.16	in AC_0
L	ExpTime UB: Thm. 7.10	PTime UB: Thm. 7.11	in AC_0 UB: Thm. 7.13
ML	2ExpTime LB: Thm. 7.15	ExpTime LB: Thm. 7.17	coNP LB: [Alviano et al. 2012, Thm. 7]
G	2ExpTime	ExpTime	coNP
FG	2ExpTime	2ExpTime LB: Thm. 6.5	coNP UB: Thm. 4.3
WG	2ExpTime	ExpTime UB: Thm. 6.2	ExpTime LB: [Calì et al. 2013, Thm. 4.1]
WFG	2ExpTime UB: Thm. 4.1	2ExpTime	ExpTime UB: Thm. 4.4

Table III: Complexity of answering atomic queries under guarded-based DTGDs. Recall that the ExpTime upper bound provided by Theorem 6.2 holds with the additional assumption that the number of body-variables is bounded.

7. ATOMIC QUERIES

We continue our complexity analysis of query answering under the respective guarded-based classes of DTGDs by concentrating on *atomic queries (CQ_1)*, i.e., CQs consisting of a single atom.

7.1. Overview

Let us first clarify that the setting where the given set of DTGDs is fixed coincides with the setting where both the set of DTGDs and the query are fixed (i.e., the data complexity). By fixing the set of DTGDs, we implicitly fix the arity of the underlying schema, and thus the size of the given atomic query. For this reason, we exclude from our complexity analysis the case where the set of DTGDs is fixed.

Table III summarizes the complexity of CQ_1 answering. Compared with the results presented in Table II, it is clear that for the formalisms that allow more than one body-atoms (i.e., all the considered classes excluding DIDs and linear DTGDs) the complexity of our problem does not change if we focus on atomic queries. However, for DIDs and linear DTGDs the complexity decreases significantly, and in fact we obtain our first tractability results. More precisely, the combined complexity decreases from 2ExpTime to ExpTime, in the case of predicates of bounded arity the complexity decreases from ExpTime to PTime, and the data complexity decreases from coNP to AC_0. The novel results of this section follow:

Upper Bounds:

(1) CQ_1 answering under linear DTGDs is in ExpTime in combined complexity (Theorem 7.10) — this is shown by exhibiting an alternating algorithm that uses polynomial space;

(2) CQ_1 answering under linear DTGDs is in PTIME if we focus on predicates of bounded arity (Theorem 7.11) — the alternating algorithm mentioned above uses only logarithmic space when the arity is fixed; and
(3) CQ_1 under linear DTGDs is in AC_0 in data complexity (Theorem 7.13) — by showing that our problem is first-order rewritable, i.e., it can be reduced to the problem of evaluating a first-order query (which depends only on the given set of DTGDs and the given query) over the input database.

Lower Bounds:

(1) CQ_1 answering under DIDs is EXPTIME-hard in combined complexity (Theorem 7.14) — this is established by simulating the behavior of an alternating linear space Turing machine;
(2) CQ_1 answering under multi-linear DTGDs is 2EXPTIME-hard in combined complexity (Theorem 7.15) — by adapting the proof of Theorem 6.4 in such a way that each disjunct of the employed UCQ can be conceived as a multi-linear DTGD that can be added in the constructed set of DTGDs, and eventually answer a single propositional atom;
(3) CQ_1 answering under DIDs is PTIME-hard, even for predicates of bounded arity (Theorem 7.16) — by simulating a logarithmic space Turing machine; and
(4) CQ_1 answering under multi-linear DTGDs is EXPTIME-hard, even if we focus on predicates of bounded arity (Theorem 7.17) — by simulating an alternating polynomial space Turing machine; in fact, the proof of this result is an adaptation of the proof of Theorem 7.15 (which in turn is an adaptation of the one of Theorem 6.4) in such a way that the (polynomially many) cells are encoded as part of the predicate name, and thus the arity becomes fixed.

Several missing complexity bounds are obtained from results of Sections 4 and 6 (see Table III). Let us clarify that the 2EXPTIME-hardness of CQ_1 answering under frontier-guarded DTGDs, in the case of predicates of bounded arity, is inherited from Theorem 6.5 since the latter holds even for queries consisting of a single propositional atom; this fact is implicit in the proof of Theorem 6.5. Finally, the following two results from the literature provide us with optimal lower bounds for the data complexity of our problem, and the entire picture of the complexity of CQ_1 answering is completed.

Inherited Results:

(1) CQ_1 answering under multi-linear DTGDs is coNP-hard in data complexity [Alviano et al. 2012, Theorem 7]; and
(2) CQ_1 answering under weakly-guarded DTGDs is EXPTIME-hard in data complexity [Calì et al. 2013, Theorem 4.1].

From the results of this section, we observe that atomic queries do not make the query answering problem easier under those classes of DTGDs that allow more than one atom in the body of a DTGD. However, the complexity in the case of DIDs and linear DTGDs significantly decreases, and we obtain several tractability results. Let us now proceed with the formal proofs of our results.

7.2. Upper Bounds

Up to now, all the upper bounds presented in the previous sections have been established by reducing our problem to the problem of reasoning (either query answering or satisfiability) under expressive computation logics such as the guarded fragment of first-order logic. However, for CQ_1 answering under linear DTGDs this is not possible. All the upper bounds that can be obtained by following the usual approach are

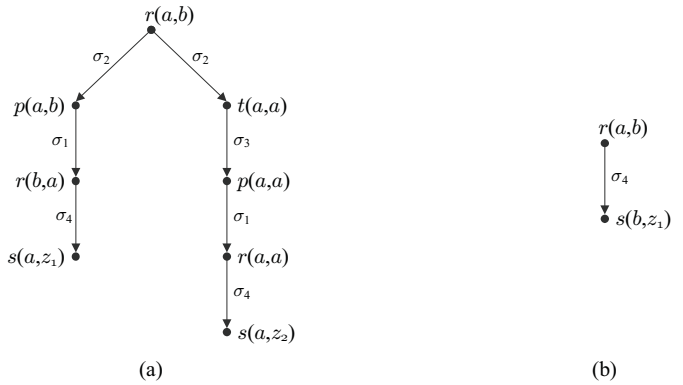

Fig. 6: Possible proof-trees of $r(a, b)$ and Σ.

not optimal, and thus more refined techniques are needed. Our plan of attack is to reduce CQ_1 answering under linear DTGDs to the problem of deciding the existence of a proof-tree, that is, a tree structure that encodes the finite part of each model of the given database w.r.t. the given set of DTGDs due to which the query is entailed, and then exhibit an alternating algorithm for deciding whether such a structure exists.

Proof-Trees. Let us start by introducing the notion of the proof-tree of an atom w.r.t. a set of DTGDs, and establish some key properties. In what follows, we consider linear DTGDs in normal form (see Section 2) in order to simplify our technical definitions and proofs.

Definition 7.1 (*Proof-tree*). Consider an atom $\underline{a}$, where $dom(\underline{a}) \subset \mathbf{C}$, and a set Σ of linear DTGDs in normal form. Let T be a labeled binary tree $(N, E, \lambda_1, \lambda_2)$, where the nodes of N are labeled by λ_1 with atoms that can be formed using predicates of $sch(\Sigma)$ and terms of $(dom(\underline{a}) \cup \mathbf{N})$, and the edges of E are labeled by $\lambda_2 : E \to \Sigma$. We say that T is a *proof-tree* of $\underline{a}$ and Σ if:

— the root is labeled by $\underline{a}$;
— for each $v \in N$, there exists $\sigma \in \Sigma$ such that each edge of the form $(u, v) \in E$ is labeled by σ, and the out-degree of v is $|head(\sigma)|$;
— for each $v \in N$, if v has a single outgoing edge (v, u) labeled by $p(\mathbf{X}) \to \exists Y\, r(\mathbf{X}, Y)$, then there is a homomorphism h such that $h(p(\mathbf{X})) = \lambda_1(v)$, and there exists $h' = h_{|\mathbf{x}} \cup \{Y \to t \mid t \in \mathbf{N},\, t \notin dom(h(p(\mathbf{X})))\}$ such that $\lambda_1(u) = h'(r(\mathbf{X}, Y))$; and
— for each $v \in N$, assuming that the outgoing edges of v are labeled by a DTGD σ without an existentially quantified variable, there is a homomorphism h such that $h(body(\sigma)) = \lambda_1(v)$, and $\{h(\underline{b}) \mid \underline{b} \in head(\sigma)\} = \{\lambda_1(u) \mid (v, u) \in E\}$.

A proof-tree T of $\underline{a}$ and Σ is *valid* w.r.t. to a CQ_1 of the form $\exists \mathbf{X}\, p(\mathbf{X})$ if, for each leaf u of T, there exists a homomorphism h such that $\lambda_1(u) = h(p(\mathbf{X}))$. □

Example 7.2. Consider the set Σ of linear DTGDs consisting of

$$\sigma_1 = p(X, Y) \to r(Y, X) \qquad \sigma_2 = r(X, Y) \to p(X, Y) \vee t(X, X)$$
$$\sigma_3 = t(X, Y) \to p(X, Y) \qquad \sigma_4 = r(X, Y) \to \exists Z\, s(Y, Z).$$

Possible proof-trees of the atom $r(a, b)$ and Σ are shown in Figure 6. In particular, tree (a) is valid w.r.t. the atomic queries $\exists X\, s(a, X)$ and $\exists X \exists Y\, s(X, Y)$, while tree (b) is valid w.r.t. $\exists X\, s(b, X)$ and $\exists X \exists Y\, s(X, Y)$. ■

The following key lemma, established in [Alviano et al. 2012, Lemma 4], shows that for CQ_1 answering under linear DTGDs one can focus on a single database atom; for the proof see Appendix D:

LEMMA 7.3. *Consider a database D, a set Σ of linear DTGDs in normal form, and a CQ_1 q. Then, $D \cup \Sigma \models q$ iff there exists $\underline{a} \in D$ such that $\{\underline{a}\} \cup \Sigma \models q$.*

Although we can focus on a single database atom, in general we have to consider infinitely many models. The key idea underlying our approach is to use "representatives" of all these models in order to be able to encode them in a single structure, namely the proof-tree defined above. This can be achieved by considering the skolemized version of the given set of DTGDs. Given a set Σ of linear DTGDs, we define F_Σ as the set of *skolem functions* $\{f_\sigma \mid \sigma \in \Sigma\}$, where the arity of each f_σ is the number of universally quantified variables of σ. Let Σ_f denote the set of rules obtained from Σ by replacing each TGD σ of the form $p(\mathbf{X}) \rightarrow \exists Y\, r(\mathbf{X}, Y)$ with the rule $p(\mathbf{X}) \rightarrow r(\mathbf{X}, f_\sigma(\mathbf{X}))$. From well-known results on skolemization the following holds: Given a database D, a set Σ of linear DTGDs in normal form, and a CQ_1 q, $D \cup \Sigma \models q$ iff $D \cup \Sigma_f \models q$. The set of *skolem terms* Γ_Σ is recursively defined as follows: each term of $\mathbf{C}$ belongs to Γ_Σ, and if $f_\sigma \in F_\Sigma$ has arity $n > 0$ and $t_1, \ldots, t_n$ are terms of Γ_Σ, then $f_\sigma(t_1, \ldots, t_n) \in \Gamma_\Sigma$. The notion of homomorphism naturally extends to atoms that contain functional terms of the form $f(t_1, \ldots, t_n)$, where each t_i is a variable of $\mathbf{V}$ or a skolem term; in particular, given a homomorphism h, $h(f(t_1, \ldots, t_n)) = f(h(t_1), \ldots, h(t_n))$. In what follows, it is more convenient to conceive models as trees. In fact, as we shall see, this will allow us to easily build the desired proof-tree of an atom and a set of DTGDs.

Definition 7.4 (ρ-*tree*). Consider a set Σ of linear DTGDs in normal form, an atom $\underline{a}$, where $dom(a) \subset \Gamma_\Sigma$, and an instance $M \in mods(\{\underline{a}\}, \Sigma_f)$. The *tree* of M, denoted $tree(M)$, is a labeled rooted tree $(N, E, \lambda_1, \lambda_2)$, where N is the node set, E is the edge set, $\lambda_1 : N \rightarrow M$ is a node labeling function, and $\lambda_2 : E \rightarrow \Sigma_f$ is an edge labeling function, such that:

— the root of $tree(M)$ is labeled by $\underline{a}$; and
— for each node $v \in N$, and rule $\rho \in \Sigma_f$ for which there exists a homomorphism h that maps $body(\rho)$ into $\lambda_1(v)$, the following holds: for each $\underline{b} \in head(\sigma)$, if $h(\underline{b}) \in M$, then there exists $u \in N$ with $\lambda_1(u) = h(\underline{b})$, $e = (v, u)$ belongs to E, and $\lambda_2(e) = \rho$.

The ρ-*tree* of M, denoted ρ-$tree(M)$, is the tree obtained from $tree(M)$ by keeping the root node v, each edge $e = (v, u)$ which is labeled by ρ, and the subtree rooted at u. Let (ρ-)$trees(\underline{a}, \Sigma_f) = \{(\rho\text{-})tree(M) \mid M \in mods(\{\underline{a}\}, \Sigma_f)\}$. By abuse of notation, given a (ρ-)tree T and a CQ_1 q, we write $T \models q$ if $\{\lambda_1(v)\}_{v \in N} \models q$. □

Example 7.5. Consider the set Σ_f consisting of

$$\rho_1 = p(X,Y) \rightarrow r(Y,X) \qquad \rho_2 = r(X,Y) \rightarrow p(X,Y) \vee t(X,X)$$
$$\rho_3 = t(X,Y) \rightarrow p(X,Y) \qquad \rho_4 = r(X,Y) \rightarrow s(Y, f(X,Y)).$$

Possible models of $\{r(a,b)\} \cup \Sigma_f$ are:

$M_1 = \{r(a,b), s(b, f(a,b)), t(a,a), p(a,a), r(a,a), s(a, f(a,a))\}$,
$M_2 = \{r(a,b), s(b, f(a,b)), p(a,b), r(b,a), t(b,b), p(b,b), r(b,b), s(a, f(b,a)), s(b, f(b,b))\}$,
$M_3 = \{r(a,b), s(b, f(a,b)), p(a,b), r(b,a), p(b,a), s(a, f(b,a))\}$.

Notice that, for each other model M of $\{r(a,b)\} \cup \Sigma_f$, it holds that $M_i \subseteq M$, for at least one $i \in [3]$; the tree T_i of M_i is depicted in Figure 7. Observe that each T_i has exactly one ρ_2-tree and one ρ_4-tree. Moreover, the shaded paths form (modulo null renaming) the proof-tree of $r(a,b)$ and Σ_f shown in Figure 6(a), which is valid w.r.t. $s(a, X)$. ■

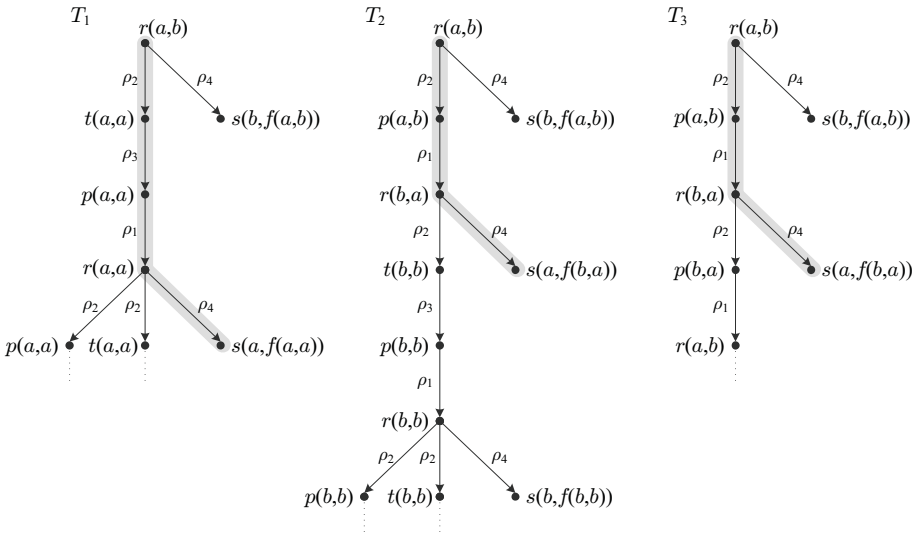

Fig. 7: Possible trees of $trees(\underline{d}, \Sigma_f)$.

Observe that the first edge of each shaded path in Figure 7 is labeled by the same rule. As shown in the next technical lemma, this is a general property that holds whenever the given database and set of DTGDs entail the given atomic query.

LEMMA 7.6. *Consider a set Σ of linear DTGDs in normal form, an atom $\underline{a}$, where $dom(\underline{a}) \subset \Gamma_\Sigma$, and a CQ$_1$ q. It holds that $\{a\} \cup \Sigma_f \models q$ iff there exists $\rho \in \Sigma_f$ such that $T \models q$, for each $T \in \rho\text{-}trees(\underline{a}, \Sigma_f)$.*

Given an atom $\underline{a}$, where $dom(a) \subset \mathbf{C}$, a set Σ of linear DTGDs, and a CQ$_1$ q, by applying recursively the property guaranteed by Lemma 7.6, one can build a proof-tree of $\underline{a}$ and Σ which is valid w.r.t. q. This is exactly the key idea underlying the proof of the next crucial result, which can be found in Appendix D.

LEMMA 7.7. *Consider a database D, a set Σ of linear DTGDs in normal form, and a CQ$_1$ q. It holds that $D \cup \Sigma \models q$ iff there exists a proof-tree of $\underline{a}$ and Σ, for some $\underline{a} \in D$, which is valid w.r.t. q.*

By Lemma 7.7, CQ$_1$ answering is equivalent to the problem of deciding whether a proof-tree exists. The latter can be tackled via an alternating algorithm.

The Algorithm. The alternating algorithm called SearchPT takes as input a database D, a set Σ of linear DTGDs, and a CQ$_1$ q, and decides whether a proof-tree of $\underline{a}$ and $\mathsf{N}(\Sigma)$, for some $\underline{a} \in D$, which is valid w.r.t. q exists. The formal definition of SearchPT is presented in Figure 8, while an example of its computation follows.

Example 7.8. Consider the database $D = \{r(a, b)\}$, the set Σ of linear DTGDs

$$\sigma_1 = r(X,Y) \rightarrow p(X) \vee t(Y,X) \qquad \sigma_2 = p(X) \rightarrow r(X,X)$$
$$\sigma_3 = t(X,Y) \rightarrow \exists Z\, s(Y,Z) \qquad \sigma_4 = s(X,Y) \rightarrow \exists Z\, s(Y,Z),$$

and the CQ$_1$ $q = \exists \mathbf{X}\, p(\mathbf{X})$. Figure 9 shows an initial part of the alternating computation of SearchPT(D, Σ, q). Observe that exactly three (i.e., maximum arity plus one) nulls appear in the shaded edge. ∎

Correctness of SearchPT follows by construction. Lemma 7.7 implies the following:

Algorithm $\mathsf{SearchPT}(D, \Sigma, q)$

Input: A database D, a set Σ of linear DTGDs, and a CQ_1 $q = \exists \mathbf{X}\, p(\mathbf{X})$.
Output: *Accept* iff there exists a "small" proof-tree of $\underline{a}$ and $\mathsf{N}(\Sigma)$, for some $\underline{a} \in D$, which is valid w.r.t. q.

(1) $\Sigma := \mathsf{N}(\Sigma)$;
(2) $N := \{z_i \in \mathbf{N}\}_{i \in [m+1]}$, where m is the maximum arity over all predicates of $sch(\Sigma)$;
(3) Guess an atom $\underline{a} \in D$;
(4) Guess to either execute step 5 or to skip to step 6;
(5) If there exists a homomorphism h such that $h(p(\mathbf{X})) = \underline{a}$, then *accept*;
(6) Guess a DTGD $\sigma \in \Sigma$ of the form $r(\mathbf{X}) \rightarrow \exists Y\, \psi_i(\mathbf{X}, Y)$ (Y may not be present), and a homomorphism h such that $h(body(\sigma)) = \underline{a}$; if there is no such σ and h, then *reject*;
(7) Universally choose each disjunct $\underline{b}$ of $head(\sigma)$ and do the following:
   (a) If Y is present in σ, then $h' := h_{|\mathbf{x}} \cup \{Y \rightarrow t \mid t \in N \text{ and } t \notin dom(h(r(\mathbf{X})))\}$; otherwise, $h' := h$;
   (b) $\underline{a} := h'(\underline{b})$ and goto step 4.

Fig. 8: The alternating algorithm SearchPT.

PROPOSITION 7.9. *Consider a database D, a set Σ of linear DTGDs, and a CQ_1 q. Then, $D \cup \Sigma \models q$ iff $\mathsf{SearchPT}(D, \Sigma, q)$ accepts.*

Complexity Upper Bounds. Equipped with the above machinery, we are now ready to establish the desired complexity upper bounds for our problem.

THEOREM 7.10. *CQ_1 answering under linear DTGDs is in* EXPTIME *in combined complexity.*

PROOF. Consider a database D, a set Σ of linear DTGDs, and a CQ_1 q. Recall that alternating PSPACE coincides with EXPTIME. Thus, by Proposition 7.9, and since SearchPT is an alternating procedure, it suffices to show that $\mathsf{SearchPT}(D, \Sigma, q)$ uses polynomial space in general. Clearly, the set of nulls N can be maintained in $O(m \log m)$ space, where m is the maximum arity over all predicates of $sch(\mathsf{N}(\Sigma))$. Furthermore, the atom $h'(\underline{b})$ can be maintained in $\mathcal{O}(m \log m + m \log n)$ space, where $n = |dom(D)|$. This is because $h'(\underline{b})$ contains at most m terms, and each term can be represented using $\log m$ bits, if is a null of N, or $\log n$ bits, if is a constant of $dom(D)$. Therefore, at each step of its computation the algorithm SearchPT uses $\mathcal{O}(m \log m + m \log n)$ space. □

We proceed to show that CQ_1 answering under linear DTGDs is in PTIME in case of predicates of bounded arity. Since alternating LOGSPACE coincides with PTIME, it suffices to show that $\mathsf{SearchPT}(D, \Sigma, q)$ uses only logarithmic space. Recall that our algorithm uses $\mathcal{O}(m \log m + m \log n)$ space, where m is the maximum arity over all predicates of $sch(\mathsf{N}(\Sigma))$. However, the existing normalization procedure (see Section 2) introduces new auxiliary predicates of unbounded arity, and thus m is not an integer constant. Thus, we need a normalization procedure that does not increase the arity of the original schema. Although it is not clear how such a normalization procedure works in general, for CQ_1 answering under linear DTGDs such a procedure can be easily defined. The key fact is that we do not need to preserve the joins among nulls since linear DTGDs can have only one body-atom, and also the query to be answered is a single atom. The normalization procedure can be found in Appendix D. With such a normalization procedure in place we get the following:

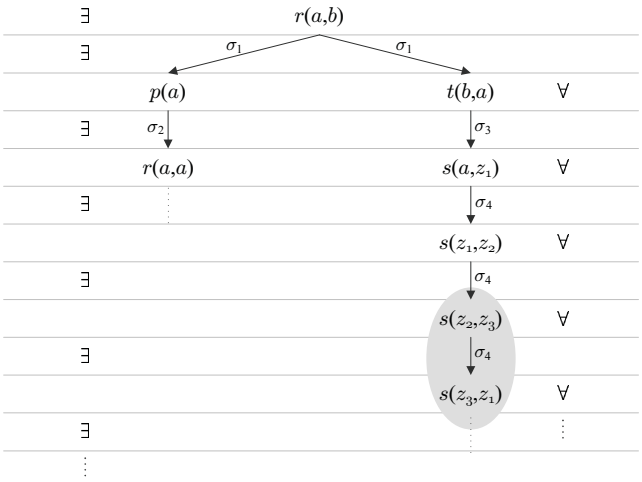

Fig. 9: Computation of the algorithm SearchPT.

THEOREM 7.11. *CQ_1 answering under linear DTGDs is in* PTIME *in case of predicates of bounded arity.*

The rest of this subsection is devoted to show that CQ_1 answering under linear DTGDs is in AC_0 in data complexity. We do this by establishing that our problem is first-order rewritable, i.e., it can be reduced to the problem of evaluating a first-order query over a database. First-order rewritability was first introduced in the context of description logics [Calvanese et al. 2007].

Consider a set Σ of linear DTGDs, and a CQ_1 q. Let C be the set of constants occurring in q, and $N = \{z_1, \ldots, z_m\}$ be a set of nulls, where m is the maximum arity over all predicates of $sch(\Sigma)$. Let $base(q, \Sigma)$ be the set of all atoms that can be formed using terms of $C \cup N$ and predicates of $sch(\Sigma)$. Let $B = \{\underline{b} \mid \underline{b} \in base(q, \Sigma) \text{ and } \{\underline{b}\} \cup \Sigma \models q\}$, and μ be a renaming substitution that maps each $z \in N$ into a distinct variable $X_z \in \mathbf{V}$. We define the first-order query q_Σ as

$$\bigvee_{\underline{b} \in B} \exists X_{z_1} \ldots \exists X_{z_m} \, \mu(\underline{b}).$$

In fact, q_Σ is a union of CQs, where each disjunct is an atomic CQ. It is easy to see that $|B| \leqslant |sch(\Sigma)| \cdot (2m)^m$. Since, by Theorem 7.10, CQ_1 answering under linear DTGDs is feasible in exponential time, we conclude that the set B, and thus the query q_Σ, can be constructed in exponential time. In what follows, we show that q_Σ is a sound and complete rewriting:

LEMMA 7.12. *Consider a set Σ of linear DTGDs in normal form, and a CQ_1 q. Then, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$.*

PROOF. Fix a database D and let $q = \exists \mathbf{X} \, p(\mathbf{X})$. By construction of q_Σ, it suffices to show that $D \cup \Sigma \models q$ iff there exists $\underline{b} \in base(q, \Sigma)$ and a homomorphism h such that $h(\underline{b}) \in D$ and $\{\underline{b}\} \cup \Sigma \models q$.

($\Rightarrow$) By Lemma 7.3, there exists an atom $\underline{a} \in D$ such that $\{\underline{a}\} \cup \Sigma \models q$. Therefore, for each $M \in mods(\{\underline{a}\}, \Sigma)$, there exists a homomorphism h_M such that $h_M(p(\mathbf{X})) \subseteq M$. Moreover, by construction, there exists $\underline{b} \in base(q, \Sigma)$ and a bijective homomorphism λ such that $\lambda(\underline{b}) = \underline{a}$; clearly, $\lambda(\underline{b}) \in D$. Let $\gamma : dom(\underline{a}) \to dom(\underline{b})$ be the substitution

$\{t \to t' \mid t' \to t \in \lambda_{|_{dom(\underline{b})}}\}$. By induction on the depth of the trees of the models, it can be shown that for each $M' \in mods(\{\underline{b}\}, \Sigma)$, there exists $M \in mods(\{\underline{a}\}, \Sigma)$ such that $\gamma(M) = M'$; thus, the homomorphism $\gamma \circ h_M$ maps $p(\mathbf{X})$ into M'.

($\Leftarrow$) We are going to show that $\{h(\underline{b})\} \cup \Sigma \models q$, which in turn implies that $D \cup \Sigma \models q$, as needed. The latter holds by Lemma 7.3 and the fact that $h(\underline{b}) \in D$. Since, by hypothesis, $\{\underline{b}\} \cup \Sigma \models q$, it suffices to show that, for each $I \in chase(\{h(\underline{b})\}, \Sigma)$, $I \in mods(\{\underline{b}\}, \Sigma)$. Fix an arbitrary instance $I \in mods(\{\underline{b}\}, \Sigma)$. Clearly, $\lambda \circ h$, where $\lambda = \{t \to t \mid t \in dom(h(\underline{b}))\}$, maps $\underline{b}$ to I. This immediately implies that $I \in mods(\{\underline{b}\}, \Sigma)$, and the claim follows. □

In the case of data complexity, the set of DTGDs Σ and the CQ_1 q are fixed, and thus q_Σ is fixed. Since the evaluation of first-order queries is feasible in AC_0 when the query is fixed [Vardi 1995], Lemma 7.12 implies the following complexity result:

THEOREM 7.13. *CQ_1 answering under linear DTGDs is in AC_0 in data complexity.*

This concludes the upper bound section for atomic queries.

7.3. Lower Bounds

We proceed now to establish the desired complexity lower bounds for CQ_1 answering under DIDs. The following theorem is obtained by simulating an alternating polynomial space Turing machine, using the ideas of the PSPACE-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al. 1984].

THEOREM 7.14. *CQ_1 answering under DIDs is* EXPTIME*-hard in combined complexity.*

PROOF. The proof is by reduction to the acceptance problem of an alternating linear space Turing machine M on input $I = a_1 \ \ldots a_{|I|}$. Let $M = (S, \Lambda, \delta, s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0, 1, \sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S \times \Lambda \to (S \times \Lambda \times \{-1, +1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, always halts, and uses exactly $n = |I|$ tape cells. Also, for each transition $(s, a) \to ((s', a', d'), (s'', a'', d''))$, we assume that $d' = d''$; the latter is a technical assumption which will allow us to represent the transitions of δ in a more convenient form (in the same way as in the PSPACE-hardness proof of the implication problem of (non-disjunctive) IDs given in [Casanova et al. 1984]). We represent configurations using a string $\Lambda^\star S \Lambda^+$, i.e., the state of the configuration is placed to the immediate left of the cursor position. In this notation, the initial configuration is $s_0 a_1 \ldots a_{|I|}$. Finally, we assume that the accepting configuration of M is $s_a \sqcup^n$.

Our goal is to construct a database D, a set Σ of DIDs, and a CQ_1 q such that $D \cup \Sigma \models q$ iff M accepts. The idea is to represent the configurations of M by strings of length $(|S| + |\Lambda|) \cdot (n + 1)$, which are stored in the predicate $conf$. Each symbol of $(S \cup \Lambda)$ will be represented by a string of length $(|S| + |\Lambda|)$ of the form $(\{0\} \cup \mathbf{N})^\star 1 (\{0\} \cup \mathbf{N})^\star$. To this end, we assume the order $s_0 < s_1 < \ldots < s_{|S|-1} < 0 < 1 < \sqcup$ on the elements of $(S \cup \Lambda)$. A symbol $x \in (S \cup \Lambda)$ that occurs at the i-th position of the above order is represented by a string $(\{0\} \cup \mathbf{N})^{i-1} 1 (\{0\} \cup \mathbf{N})^{(|S|+|\Lambda|)-i}$. For example, a possible encoding for s_0 is $1, 0, \ldots, 0$, while for $\sqcup$ is $0, \ldots, 0, 1$. Now, given a configuration $C = x_1, x_2, \ldots, x_{n+1}$, the encoding of C is a string of length $(|S| + |\Lambda|) \cdot (n + 1)$, where at its i-th position a possible encoding of x_i occurs. Formally, assuming a total order on the elements of $(S \cup \Lambda) \times [n + 1]$, where $(x, i) < (y, j)$ iff $(i < j)$ or $(i = j$ and $x < y)$, with $\#_{(x,i)}$ being the position of (x, i) in this order, the encoding of a configuration C contains 1 at po-

sition $\#_{(x,i)}$ iff the i-th symbol in C is x. We proceed with the construction of D, Σ and q.

The Database D. D consists of a single atom which represents the initial configuration $s_0 a_1 \dots a_{|I|}$ of M. Given two elements $x, y \in \Lambda$, $f(x,y) = 1$ if $x = y$; otherwise, $f(x,y) = 0$. The database atom is defined as $\{conf(\mathbf{t})\}$, where $\mathbf{t}$ is the binary tuple:

$$\underbrace{1,\overbrace{0,\dots,0}^{|S|-1},0,0,0}_{s_0},\underbrace{\overbrace{0,\dots,0}^{|S|},f(a_1,0),f(a_1,1),f(a_1,\sqcup)}_{a_1},\dots,\underbrace{\overbrace{0,\dots,0}^{|S|},f(a_{|I|},0),f(a_{|I|},1),f(a_{|I|},\sqcup)}_{a_{|I|}}.$$

Clearly, $\mathbf{t}$ is obtained by replacing each symbol occurring in the initial configuration of M, by its (unique) encoding that contains only symbols of $\{0,1\}$.

The Set Σ. With Σ we encode the transitions of δ. These transitions can be described by expressions of the form $x_1y_1z_1 \rightarrow x_2y_2z_2; x_3y_3z_3$ with $x_i, y_i, z_i \in (S \cup \Lambda)$, for each $i \in \{1,2,3\}$. For example, the transition $(s,a) \rightarrow ((s',a',-1),(s'',a'',-1))$ corresponds to $xsa \rightarrow s'xa'; s''xa''$, for each $x \in \Lambda$, while the transition $(s,a) \rightarrow ((s',a',+1),(s'',a'',+1))$ corresponds to $sax \rightarrow a's'x; a''s''x$, for each $x \in \Lambda$. Let T_δ be all such expressions corresponding to transitions of δ.

Consider an expression $\tau \in T_\delta$ of the form $x_1y_1z_1 \rightarrow x_2y_2z_2; x_3y_3z_3$. For each $i \in \{1,2,3\}$ and $j \in [n-1]$, we define the atom $g_i^j(\tau)$ as $conf(\mathbf{t})$, where $\mathbf{t} \in \mathbf{V}^{ar}$ is as follows:

— at positions $\#_{(x_i,j)}$, $\#_{(y_i,j+1)}$ and $\#_{(z_i,j+2)}$ of $\mathbf{t}$ the variables X, Y and Z occur, respectively;
— for each $(x,\ell) \in (\Lambda \times \{1,\dots,j-1,j+3,\dots,n+1\})$, at position $\#_{(x,\ell)}$ of $\mathbf{t}$ the variable X_ℓ occurs; and
— at each position of $\mathbf{t}$ not considered above a distinct variable occurs; these variables form the set $\mathbf{Y}_i$.

Having the above atoms in place, we are now ready to define the set Σ of DIDs. For each $\tau = x_1y_1z_1 \rightarrow x_1y_2z_2; x_3y_3z_3$ of T_δ, where $s \in S$ is the state occurring in $x_1y_1z_1$, and for each $j \in [n-1]$:

— If $s \in S_\forall$, then Σ includes the DID

$$g_1^j(\tau) \;\rightarrow\; \exists \mathbf{Y}_2\, g_2^j(\tau) \vee \exists \mathbf{Y}_3\, g_3^j(\tau).$$

— If $s \in S_\exists$, then Σ includes the two (non-disjunctive) IDs

$$g_1^j(\tau) \;\rightarrow\; \exists \mathbf{Y}_2\, g_2^j(\tau) \qquad \text{and} \qquad g_1^j(\tau) \;\rightarrow\; \exists \mathbf{Y}_3\, g_3^j(\tau).$$

This completes the construction of Σ.

The CQ$_1$ q. The query q is defined as $\exists \mathbf{X}\, conf(\mathbf{t})$, where $\mathbf{t} \in (\{1\} \cup \mathbf{X})^{ar}$ is as follows:

— At positions $\#_{(s_a,1)}, \#_{(\sqcup,2)}, \dots, \#_{(\sqcup,n+1)}$ the constant 1 occurs; and
— At all the other positions a distinct variable of $\mathbf{X}$ occurs.

The construction of q is now complete.

By construction, whenever a transition $\tau \in \delta$ takes place, at least one of the DIDs of Σ will propagate the constant 1 in such a way that the derived atom will represent the successor configuration. If the machine thus reaches the accepting configuration, this implies that there exists a sequence of DIDs that will derive the atom corresponding to that accepting configuration, namely the CQ$_1$ q. Conversely, if $D \cup \Sigma \models q$, then there

exists a sequence of DIDs as above, and the sequence of corresponding transitions in δ yields an accepting computation of M. □

We now focus on multi-linear DTGDs, and show that query answering remains 2EXPTIME-hard, even if we consider atomic queries. Let D, Σ and Q be the database, the set of DIDs, and the UCQ employed in the proof of Theorem 6.4, where it is shown that ACQ answering under DIDs is 2EXPTIME-hard. We can construct in polynomial time a set Σ' of multi-linear DTGDs, which depends on Σ and Q, such that $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models q$, where $q = p^\star$ with $p^\star$ being an auxiliary 0-ary predicate not occurring in $sch(\Sigma)$; for the formal construction see Appendix D. In fact, our intention is to obtain Σ' by adding to Σ the TGD $\varphi \rightarrow p^\star$, for each disjunct φ of Q. The next result follows:

THEOREM 7.15. *CQ_1 answering under multi-linear DTGDs is 2EXPTIME-hard in combined complexity.*

In the case of predicates of bounded arity, we can show that atomic query answering under DIDs is PTIME-hard; in fact, this is true even if we focus on unary predicates. The proof, which can be found in Appendix D, is by reduction from the acceptance problem of an alternating logarithmic space Turing machine.

THEOREM 7.16. *CQ_1 answering under DIDs is PTIME-hard, even for unary predicates.*

The last lower bound that we establish for CQ_1 answering is the EXPTIME-hardness when we focus on multi-linear DTGDs and predicates of bounded arity. This result can be obtained by adapting the proof of Theorem 7.15. Since we have to simulate an alternating polynomial space Turing machine (and not an exponential space one), the polynomially many cells can be encoded in the predicates, and thus the arity becomes fixed. The proof can be found in Appendix D.

THEOREM 7.17. *CQ_1 answering under multi-linear DTGDs is EXPTIME-hard, even for predicates of bounded arity.*

8. QUERYING DESCRIPTION LOGIC KNOWLEDGE BASES

Description logics (DLs) [Baader et al. 2003] have been playing a prominent role in ontological reasoning. Most DLs are decidable fragments of first-order logic, based on concepts (classes of objects) and roles (binary relations on objects). Several variants of DLs have been proposed, where a central issue is the trade-off between the expressive power and the computational complexity of the reasoning services such as query answering. In this section, we show that our techniques and results on guarded DTGDs can be used as a generic tool for establishing results on query answering under several central DLs. To this end, we focus on the core DL $\mathcal{ALCHI}$.

8.1. The Description Logic $\mathcal{ALCHI}$

$\mathcal{ALC}$ has been introduced in [Schmidt-Schauß and Smolka 1991], and is at the basis of many expressive DLs. By enriching $\mathcal{ALC}$ with role hierarchies ($\mathcal{H}$) and inverse roles ($\mathcal{I}$) we obtain the expressive DL $\mathcal{ALCHI}$. Let $\mathbf{N_C}$ and $\mathbf{N_R}$ be disjoint countably infinite sets of concept and role names, respectively. The set of $\mathcal{ALCHI}$-concepts is the smallest set such that: (1) every concept name $A \in \mathbf{N_C}$, as well as $\top$ and $\bot$, are $\mathcal{ALCHI}$-concepts; and (2) if C and D are $\mathcal{ALCHI}$-concepts and $R \in \mathbf{N_R}$, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\exists R^-.C$, $\forall R.C$ and $\forall R^-.C$ are $\mathcal{ALCHI}$-concepts. A DL *knowledge base (KB)* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ represents the domain of interest in terms of two parts, a *terminological box (TBox)* $\mathcal{T}$, specifying the intensional knowledge, and an *assertional box (ABox)* $\mathcal{A}$, asserting the extensional knowledge. In an $\mathcal{ALCHI}$ KB the TBox is a finite set of concept inclusions

of the form $B \sqsubseteq C$, where B and C are $\mathcal{ALCHI}$-concepts, as well as a finite set of role inclusions of the form $R \sqsubseteq S$, where R and S are (possibly inverse) roles. The ABox consists of a finite set of assertions $C(a)$ and $R(a, b)$, where a and b are individuals of $\mathbf{C}$, C is an $\mathcal{ALCHI}$-concept and R is a role name. The semantics of $\mathcal{ALCHI}$ is given by interpretations in the usual way; see Appendix E.

8.2. Generic Complexity Results

By exploiting our techniques and results, we can establish the following result about (BTW)UCQ answering; $\mathcal{L}_1 \subseteq \mathcal{L}_2$ means that $\mathcal{L}_1$ is a subformalism of $\mathcal{L}_2$:

THEOREM 8.1. *Consider a DL $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{ALCHI}$ and $\mathcal{L}$ is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A \qquad R \sqsubseteq S \qquad R^- \sqsubseteq S,$$

*where A, A_1, A_2, A_3 are concept names and R, S are role names. (BTW)UCQ answering under $\mathcal{L}$ is 2*EXPTIME*-complete in combined complexity, even when the TBox is fixed.*

An analogous result for AUCQ answering can be also established:

THEOREM 8.2. *Consider a DL $\mathcal{L}$ such that $\mathcal{L} \subseteq \mathcal{ALCHI}$ and $\mathcal{L}$ is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A,$$

where A, A_1, A_2, A_3 are concept names. Then, AUCQ answering under $\mathcal{L}$ is EXPTIME*-complete in combined complexity, even when the TBox is fixed.*

The upper bounds for both theorems are obtained by reducing query answering under $\mathcal{ALCHI}$ to query answering under guarded DTGDs. The desired lower bounds are inherited from Theorems 4.6, 5.1 and 6.6. In fact, the sets of DIDs employed in the proofs of the above theorems can be rewritten as DL axioms; see Appendix E.

A Case Study. We conclude this section by briefly discussing which existing DLs benefit from our generic Theorems 8.1 and 8.2. Clearly, $\mathcal{ALCHI}$ itself is one of those DLs since it is powerful enough to express all the axioms listed in our theorems. Other DLs are *DL-Lite*$^{\mathcal{H}}_{bool}$ [Artale et al. 2009], one of the most expressive languages of the DL-Lite family, and $\mathcal{ELUI}$, i.e., the extension of the well-known DL $\mathcal{EL}$ with union of concepts ($\mathcal{U}$) and inverse roles ($\mathcal{I}$). The 2EXPTIME-completeness of UCQ answering under $\mathcal{ALCHI}$ is actually known [Lutz 2008]. However, the EXPTIME-completeness of AUCQ answering under $\mathcal{ALCHI}$ (even in the case of a fixed TBox), as well as the inherited results for *DL-Lite*$^{\mathcal{H}}_{bool}$ and $\mathcal{ELUI}$ are novel.

9. CONCLUSIONS

In this paper, we have studied in depth the complexity of (U)CQ answering under the main guarded-based classes of DTGDs that can be found in the literature. Interestingly, the problem under consideration is 2EXPTIME-hard even in the case of a fixed set of DIDs, that is, the simplest guarded-based class of DTGDs. With the aim of reducing the complexity of our problem, we considered three key subclasses of (U)CQs, namely (U)CQs of bounded hypertree-width, (U)CQs of bounded treewidth and acyclic (U)CQs. Our investigation shows that the above query languages do not have the expected positive impact on our problem. Finally, towards the identification of tractable cases, we considered atomic queries. We show that for DIDs and linear DTGDs the complexity significantly decreases. Our techniques and results can be used as a generic tool for establishing complexity results for query answering under key DLs.

REFERENCES

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

Mario Alviano, Wolfgang Faber, Nicola Leone, and Marco Manna. 2012. Disjunctive Datalog with existential quantifiers: Semantics, decidability, and complexity issues. *Theory and Practice of Logic Programming* 12, 4-5 (2012), 701–718.

Hajnal Andréka, Johan van Benthem, and István Németi. 1998. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic* 27 (1998), 217–274.

Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. 2009. The DL-Lite Family and Relations. *Journal of Artificial Intelligence Research* 36 (2009), 1–69.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011a. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175, 9-10 (2011), 1620–1654.

Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. 2011b. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 712–717.

Vince Bárány, Georg Gottlob, and Martin Otto. 2014. Querying the Guarded Fragment. *Logical Methods in Computer Science* 10, 2 (2014).

Vince Bárány, Balder ten Cate, and Luc Segoufin. 2015. Guarded Negation. *Journal of the ACM* 62, 3 (2015), 22.

Catriel Beeri and Moshe Y. Vardi. 1981. The Implication Problem for Data Dependencies. In *Proceedings of the 8th International Colloquium on Automata, Languages and Programming*. 73–85.

Catriel Beeri and Moshe Y. Vardi. 1984. A Proof Procedure for Data Dependencies. *Journal of the ACM* 31, 4 (1984), 718–741.

Pierre Bourhis, Michael Morak, and Andreas Pieris. 2013. The Impact of Disjunction on Query Answering Under Guarded-Based Existential Rules. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. 796–802.

Pierre Bourhis, Michael Morak, and Andreas Pieris. 2014. Towards Efficient Reasoning Under Guarded-Based Disjunctive Existential Rules. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science, Part I*. 99–110.

Andrea Calì, Georg Gottlob, and Michael Kifer. 2008. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*. 70–80.

Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *Journal of Artificial Intelligence Research* 48 (2013), 115–174.

Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2012a. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14 (2012), 57–83.

Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. 2010. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*. 228–242.

Andrea Calì, Georg Gottlob, and Andreas Pieris. 2012b. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193 (2012), 87–128.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning* 39, 3 (2007), 385–429.

Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2013. Data complexity of query answering in description logics. *Artificial Intelligence* 195 (2013), 335–360.

Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. 1984. Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences* 28, 1 (1984), 29–59.

Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1990. *Logic Programming and Databases*. Springer.

Chandra Chekuri and Anand Rajaraman. 2000. Conjunctive query containment revisited. *Theoretical Computer Science* 239, 2 (2000), 211–229.

Bruno Courcelle. 1989. The Monadic Second-Order Logic of Graphs, II: Infinite Graphs of Bounded Width. *Mathematical Systems Theory* 21, 4 (1989), 187–221.

Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, and Ulrike Sattler. 2008. OWL 2: The next step for OWL. *Journal of Web Semantics* 6, 4 (2008), 309–322.

Alin Deutsch, Alan Nash, and Jeff B. Remmel. 2008. The Chase Revisisted. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 149–158.

Alin Deutsch and Val Tannen. 2003. Reformulation of XML Queries and Constraints. In *Proceedings of the 9th International Conference on Database Theory*. 225–241.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. 1994. Deduction in Concept Languages: From Subsumption to Instance Checking. *Journal of Logic and Computation* 4, 4 (1994), 423–452.

Thomas Eiter, Georg Gottlob, and Heikki Mannila. 1997. Disjunctive Datalog. *ACM Transactions on Database Systems* 22, 3 (1997), 364–418.

Thomas Eiter, Magdalena Ortiz, and Mantas Simkus. 2012. Conjunctive query answering in the description logic SH using knots. *J. Comput. System Sci.* 78, 1 (2012), 47–85.

Ronald Fagin. 2007. Inverting schema mappings. *ACM Transactions on Database Systems* 32, 4 (2007).

Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: Semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2008. Quasi-inverses of schema mappings. *ACM Transactions on Database Systems* 33, 2 (2008).

Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences* 64, 3 (2002), 579–627.

Georg Gottlob, Marco Manna, Michael Morak, and Andreas Pieris. 2012. On the Complexity of Ontological Reasoning under Disjunctive Existential Rules. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*. 1–18.

Erich Grädel. 1999. On The Restraining Power of Guards. *The Journal of Symbolic Logic* 64, 4 (1999), 1719–1742.

David S. Johnson and Anthony C. Klug. 1984. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences* 28, 1 (1984), 167–189.

Markus Krötzsch and Sebastian Rudolph. 2011. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 963–968.

Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. 2012. Efficiently Computable Datalog$^{\exists}$ Programs. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*.

Carsten Lutz. 2008. The Complexity of Conjunctive Query Answering in Expressive Description Logics. In *Proceedings of the 4th International Joint Conference on Automated Reasoning*. 179–193.

Christos H. Papadimitriou. 1994. *Computational Complexity*. Addison-Wesley.

Sebastian Rudolph and Birte Glimm. 2010. Nominals, Inverses, Counting, and Conjunctive Queries or: Why Infinity is your Friend! *Journal of Artificial Intelligence Research* 39 (2010), 429–481.

Andrea Schaerf. 1993. On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification. *Journal of Intelligent Information Systems* 2, 3 (1993), 265–278.

Manfred Schmidt-Schauß and Gert Smolka. 1991. Attributive Concept Descriptions with Complements. *Artificial Intelligence* 48, 1 (1991), 1–26.

Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. 2011. Consequence-Based Reasoning beyond Horn Ontologies. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. 1093–1098.

Michaël Thomazo, Jean-François Baget, Marie-Laure Mugnier, and Sebastian Rudolph. 2012. A Generic Querying Algorithm for Greedy Sets of Existential Rules. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*.

Jeffrey D. Ullman. 1987. Database Theory: Past and Future. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 1–10.

Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*. 137–146.

Moshe Y. Vardi. 1995. On the Complexity of Bounded-Variable Queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 266–276.

Online Appendix to: Guarded-Based Disjunctive Tuple-Generating Dependencies

PIERRE BOURHIS, CNRS CRIStAL, University of Lille 1 and INRIA Lille
MARCO MANNA, University of Calabria
MICHAEL MORAK, Vienna University of Technology
ANDREAS PIERIS, University of Edinburgh

A. NORMAL FORM OF DTGDS

A set Σ of DTGDs is in normal form if each $\sigma \in \Sigma$ is of the form

$$\varphi(\mathbf{X}) \rightarrow p(\mathbf{X}) \qquad \text{or} \qquad \varphi(\mathbf{X}) \rightarrow \exists Y\, p(\mathbf{X}, Y) \qquad \text{or} \qquad \varphi(\mathbf{X}) \rightarrow p_1(\mathbf{X}) \vee p_2(\mathbf{X}).$$

Every set Σ of DTGDs can be transformed in logarithmic space into a set $\mathsf{N}(\Sigma)$ in normal form such that, for every database D and UCQ Q, $D \cup \Sigma \models Q$ iff $D \cup \mathsf{N}(\Sigma) \models Q$. The normalization procedure follows.

Consider a DTGD σ. First, we construct the set $\mathsf{N}_1(\sigma)$ by exhaustively applying the following two replacement rules, starting from σ, until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

(1) A DTGD of the form $\varphi(\mathbf{X}) \rightarrow \bigvee_{i=1}^{n} \exists \mathbf{Y}_i\, \psi_i(\mathbf{X}, \mathbf{Y}_i)$ is replaced by

$$\begin{aligned} \varphi(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1\, p_1^\star(\mathbf{X}, \mathbf{Y}_1) \vee \exists \mathbf{Y}_{2..n}\, p_{2..n}^\star(\mathbf{X}, \mathbf{Y}_{2..n}) \\ p_1^\star(\mathbf{X}, \mathbf{Y}_1) &\rightarrow \psi_1(\mathbf{X}, \mathbf{Y}_1) \\ p_{2..n}^\star(\mathbf{X}, \mathbf{Y}_{2..n}) &\rightarrow \bigvee_{i=2}^{n} \psi_i(\mathbf{X}, \mathbf{Y}_i), \end{aligned}$$

where $\mathbf{Y}_{2..n} = \cup_{i=2}^{n} \mathbf{Y}_i$, $p_1^\star$ is an $|\mathbf{X} \cup \mathbf{Y}_1|$-ary predicate and $p_{2..n}^\star$ is an $|\mathbf{X} \cup \mathbf{Y}_{2..n}|$-ary predicate; both $p_1^\star$ and $p_{2..n}^\star$ are auxiliary predicates not introduced so far.

(2) A TGD of the form $\varphi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, p_1(\mathbf{X}, \mathbf{Y}_1), \ldots, p_n(\mathbf{X}, \mathbf{Y}_n)$ is replaced by

$$\begin{aligned} \varphi(\mathbf{X}) &\rightarrow \exists \mathbf{Y}\, p^\star(\mathbf{X}, \mathbf{Y}) \\ p^\star(\mathbf{X}, \mathbf{Y}) &\rightarrow p_1(\mathbf{X}, \mathbf{Y}_1) \\ &\vdots \\ p^\star(\mathbf{X}, \mathbf{Y}) &\rightarrow p_n(\mathbf{X}, \mathbf{Y}_n), \end{aligned}$$

where $p^\star$ is an $|\mathbf{X} \cup \mathbf{Y}|$-ary auxiliary predicate not introduced so far.

Now, from $\mathsf{N}_1(\sigma)$, we obtain $\mathsf{N}(\sigma)$ by applying the following: for each $\sigma' \in \mathsf{N}_1(\sigma)$, assuming that $\mathit{frontier}(\sigma') = \mathbf{X}$ and $\{Y_1, \ldots, Y_n\}$ are the existentially quantified variables of

DOI: http://dx.doi.org/10.1145/0000000.0000000

σ', σ' is replaced by

$$\begin{aligned}
body(\sigma') &\rightarrow \exists Y_1\, p_1^{\sigma'}(\mathbf{X}, Y_1)\\
p_1^{\sigma'}(\mathbf{X}, Y_1) &\rightarrow \exists Y_2\, p_2^{\sigma'}(\mathbf{X}, Y_1, Y_2)\\
p_2^{\sigma'}(\mathbf{X}, Y_1, Y_2) &\rightarrow \exists Y_3\, p_3^{\sigma'}(\mathbf{X}, Y_1, Y_2, Y_3)\\
&\vdots\\
p_{n-1}^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_{n-1}) &\rightarrow \exists Y_n\, p_n^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_n)\\
p_n^{\sigma'}(\mathbf{X}, Y_1, \ldots, Y_n) &\rightarrow head(\sigma),
\end{aligned}$$

where, for each $i \in [n]$, $p_i^{\sigma'}$ is an $(|\mathbf{X}| + i)$-ary auxiliary predicate not introduced so far. Finally, given a set Σ of DTGDs, $\mathsf{N}(\Sigma)$ is defined as $\bigcup_{\sigma\in\Sigma} \mathsf{N}(\sigma)$.

B. PROOFS FROM SECTION 4

B.1. Proof of Theorem 4.1

We provide a polynomial-time reduction to the problem of deciding whether a GNFO sentence is unsatisfiable, which in turn is in 2ExpTime [Bárány et al. 2015]. Our reduction consists of two steps: (1) first, we reduce our problem to the problem of UCQ answering under frontier-guarded DTGDs; and (2) we show how the constructed set of frontier-guarded DTGDs can be equivalently rewritten as a GNFO sentence. Consider a database D, a set Σ of weakly-frontier-guarded DTGDs, and a UCQ Q. Our reduction follows:

Step 1. The key idea is to exploit the fact that whenever the (disjunctive) chase applies a DTGD, by definition, in every model the *non-affected variables* (i.e., variables occurring at the non-affected positions) are mapped to constants of $\mathbf{C}$. It is therefore possible to partially ground the DTGDs in such a way that the non-affected variables are replaced by all possible combinations of constants, yielding an equivalent set of frontier-guarded DTGDs. However, such a transformation would be exponential in the worst case. To overcome this difficulty we adapt a technique proposed in [Baget et al. 2011b] that allows us to simulate partial groundings with only a polynomial blow-up.

Roughly speaking, instead of explicitly replacing the non-affected variables by constants, we extend each predicate of the underlying schema in such a way that it contains all m constants of the database in its first m positions, and a grounding of its n non-affected variables in the next n positions. Using a set of $|sch(\Sigma)| \cdot n \cdot m$ inclusion dependencies, each propagating one of the constants to one of the non-affected variables, we can simulate the partial grounding as desired. Using this procedure we can convert a weakly-frontier-guarded set of DTGDs into a set of frontier-guarded DTGDs in polynomial time. Let us now formalize the above informal description.

Given a DTGD $\sigma \in \Sigma$, $nav(\sigma)$ denotes the set of variables occurring in at least one position of $nonaffected(\Sigma)$. Fix also a bijection $\#_\sigma : nav(\sigma) \rightarrow [|nav(\sigma)|]$. Let $C_1, \ldots, C_m$ and $N_1, \ldots, N_n$ be variables of $\mathbf{V}$ not occurring in Σ, where m is the number of constants $c_1, \ldots, c_m$ appearing in D, and $n = \max_{\sigma\in\Sigma}\{|nav(\sigma)|\}$. Given a term t, let $\tau_\sigma(t) = N_{\#_\sigma(t)}$ if $t \in nav(\sigma)$; otherwise, let $\tau_\sigma(t) = t$. The mapping τ_σ is extended to atoms as follows:

$$\tau_\sigma(p(t_1, \ldots, t_k)) \;=\; p(C_1, \ldots, C_m, N_1, \ldots, N_n, \tau_\sigma(t_1), \ldots, \tau_\sigma(t_k)).$$

Notice that τ_σ can naturally be defined for sets of atoms. We now define the function τ, mapping DTGDs of Σ to frontier-guarded DTGDs, as follows:

$$\tau(\sigma) \;=\; \tau_\sigma(body(\sigma)) \rightarrow \tau_\sigma(head(\sigma)).$$

Intuitively, the variables $C_1, \ldots, C_m$ will permanently hold the constants of $dom(D)$, while the variables $N_1, \ldots, N_n$ will be assigned a combination of those constants. We

define the set Σ' of frontier-guarded DTGDs as the set $\{\tau(\sigma) \mid \sigma \in \Sigma\} \cup \Sigma_{\mathsf{C}}$, where Σ_{C} consists of the following IDs: $\bigcup_{p \in sch(\Sigma), i \in [m], j \in [n]} \{\sigma^p_{i,j}\}$, with

$$\sigma^p_{i,j} \;=\; p(C_1, \ldots, C_m, N_1, \ldots, N_n, X_1, \ldots, X_k) \rightarrow p(C_1, \ldots, C_m, N^\star_1, \ldots, N^\star_n, X_1, \ldots, X_k),$$

where $k = arity(p)$, $N^\star_j = C_i$, and $N^\star_k = N_k$ for each $k \neq j$. Roughly, Σ_{C} is responsible for generating all the possible combinations of the constants occurring in D. Finally, let $D' = \{\tau'(\underline{a}) \mid \underline{a} \in D\}$ and $Q' = \{\tau'(\underline{a}) \mid \underline{a} \in Q\}$, where

$$\tau'(p(t_1, \ldots, t_k)) \;=\; p(c_1, \ldots, c_m, \underbrace{c_1, \ldots, c_1}_{n}, t_1, \ldots, t_k).$$

Let us clarify that the selection of the constant c_1 at positions $m+1, \ldots, m+n$ is an arbitrary one since, due to the IDs of Σ_{C}, all the combinations of constants of $dom(D)$ will eventually appear at those positions during the construction of the chase. By construction, $D \cup \Sigma \models Q$ iff $D' \cup \Sigma' \models Q'$; this completes the first step of our reduction.

Step 2. Observe that a frontier-guarded DTGD σ of the form $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y}))$ can be equivalently rewritten as the sentence $\Phi_\sigma = \neg(\exists \mathbf{X}(\varphi(\mathbf{X}) \wedge \neg \exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})))$, which falls in GNFO since all the free variables of $\exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})$, that is, the variables of $\mathbf{X}$, appear in the frontier-guard of $\varphi(\mathbf{X})$. Moreover, given a CQ q, $\neg q$ trivially falls in GNFO since in q there are no free variables. From the above discussion, we conclude that the sentence $\Psi_{D', \Sigma', Q'} = (D' \wedge \bigwedge_{\sigma \in \Sigma'} \Phi_\sigma \wedge \bigwedge_{q \in Q'} \neg q)$ falls in GNFO. The claim follows since $D' \cup \Sigma' \models Q'$ iff $\Psi_{D', \Sigma', Q'}$ is unsatisfiable.

B.2. Proof of Lemma 4.2

Consider a database D, a set Σ of frontier-guarded DTGDs, and a UCQ Q. For each $\sigma \in \Sigma$ of the form $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}', \mathbf{Y}))$, where $\mathbf{X}' \subseteq \mathbf{X}$, we define

$$\tau_1(\sigma) \;=\; \forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow p_\sigma(\mathbf{X}')) \qquad \text{and} \qquad \tau_2(\sigma) \;=\; \forall \mathbf{X}'(p_\sigma(\mathbf{X}') \rightarrow \exists \mathbf{Y}\, \psi(\mathbf{X}', \mathbf{Y})),$$

where p_σ is an auxiliary $|\mathbf{X}'|$-ary predicate not occurring in $sch(\Sigma)$. Assuming that $g_\sigma(\mathbf{X}')$ is the frontier-guard of σ, $\tau_1(\sigma)$ can be equivalently rewritten as follows, where p'_σ is an auxiliary $|\mathbf{X}'|$-ary predicate:

$$\begin{aligned}
\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow p_\sigma(\mathbf{X}')) \;&\equiv\; \neg \exists \mathbf{X}(\varphi(\mathbf{X}) \wedge \neg p_\sigma(\mathbf{X}')) \\
&\equiv\; (\neg \exists \mathbf{X}(\varphi(\mathbf{X}) \wedge p'_\sigma(\mathbf{X}'))) \wedge (\forall \mathbf{X}'(g_\sigma(\mathbf{X}') \wedge \neg p_\sigma(\mathbf{X}') \rightarrow p'_\sigma(\mathbf{X}'))) \\
&\equiv\; (\underbrace{\neg \exists \mathbf{X}(\varphi(\mathbf{X}) \wedge p'_\sigma(\mathbf{X}'))}_{\Phi^1_\sigma}) \wedge (\underbrace{\forall \mathbf{X}'(g_\sigma(\mathbf{X}') \rightarrow p_\sigma(\mathbf{X}') \vee p'_\sigma(\mathbf{X}'))}_{\Phi^2_\sigma}).
\end{aligned}$$

We define the formulas:

$$\Psi^1_\Sigma \;=\; \left(\neg \bigvee_{\sigma \in \Sigma} \Phi^1_\sigma\right) \qquad \text{and} \qquad \Psi^2_\Sigma \;=\; \left(\bigwedge_{\sigma \in \Sigma} (\tau_2(\sigma) \wedge \Phi^2_\Sigma)\right).$$

It is not difficult to see that

$$D \cup \Sigma \models Q \qquad \text{iff} \qquad \left(D \wedge \Psi^1_\Sigma \wedge \Psi^2_\Sigma\right) \models Q \qquad \text{iff} \qquad \left(D \wedge \Psi^2_\Sigma\right) \models \left(Q \vee \neg \Psi^1_\Sigma\right).$$

Ψ^2_Σ is a set of guarded DTGDs, and thus is equivalent to a GFO sentence Γ^2_Σ; hence, $(D \wedge \Gamma^2_\Sigma)$ is a GFO sentence. Since $(Q \vee \neg \Psi^1_\Sigma)$ is a UCQ, the claim follows.

B.3. Proof of Theorem 4.6

The proof is via a reduction from the non-acceptance problem of an alternating exponential space Turing machine M on the empty input (as in the proof of the previous result). Instead of giving the complete reduction, we are going to explain how

the reduction employed in the proof of Theorem 4.5 can be adapted; notice that the database D remains the same.

The Set Σ. The part of Σ that is affected are the TGDs that construct the configuration trees. The new set of DTGDs is as follows:

$$\begin{aligned}
conf_x(X) &\rightarrow ctnode(X), \text{ for each } x \in \{\exists, \forall\},\\
ctnode(X) &\rightarrow \exists Y_L \exists Y_R\, child_L(X, Y_L), ctnode(Y_L), child_R(X, Y_R), ctnode(Y_R),\\
child_x(X, Y) &\rightarrow child(X, Y), \text{ for each } x \in \{L, R\},\\
ctnode(X) &\rightarrow \bigvee_{\lambda \in \Lambda} \lambda(X),\\
ctnode(X) &\rightarrow cursor(X) \vee notcursor(X),\\
cursor(X) &\rightarrow \exists Y\, state(X, Y),\\
state(X, Y) &\rightarrow \exists Z\, state(Y, Z) \vee end(Y).
\end{aligned}$$

Clearly, Σ does not depend on M, $\mathsf{N}(\Sigma)$ is a set of DIDs, and $sch(\mathsf{N}(\Sigma))$ consists of unary and binary predicates.

The UCQ Q. Recall that the query Q, employed in the proof of Theorem 4.5, makes use of the predicate $level_0$, in order to ask whether a node is the root of a configuration tree, and of the predicate s, where $s \in S$, in order to ask whether a cell is marked with the state s of M. However, those predicates are not available anymore, and the query Q is modified as follows:

— An atom $level_0(X)$, occurring in Q, is replaced by the formula

$$ctnode(X) \wedge (conf_\exists(X) \vee conf_\forall(X)),$$

which asks whether X is a node of a configuration tree, and at the same time represents a configuration.

— We assume a fixed order on S, and let $\#_s$ be the position of $s \in S$ in this order. The length of the state-chain which encodes s is $\#_s$. We use $state^i(X, Y)$ as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\, (state(X, Z_1) \wedge \ldots \wedge state(Z_{i-1}, Y))$, and we define the subquery:

$$state[i](X) \equiv \exists Y \left(state^i(X, Y) \wedge end(Y)\right),$$

which asks whether there exists a state-chain of length i. An atom $s(X)$, where $s \in S$, occurring in Q, is simply replaced by the atom $state[\#_s](X)$.

It should not be forgotten that we need to guarantee that each state-chain encodes a valid state. To this aim, we add another disjunct Q_{length} in Q, which is defined as

$$\bigvee_{x \in \{\exists, \forall\}} \exists X \exists Y \exists Z \left(ctnode(X) \wedge conf_x(X) \wedge child^n(X, Y) \wedge state^{|S|+1}(Y, Z)\right).$$

The definition of Q is now complete. This concludes our proof.

B.4. Proof of Lemma 4.7

We assume, w.l.o.g., that the CQs occurring in Q do not have variables in common. In the rest of the proof, let t and f be constants of $\mathbf{C}$ not occurring in D. Given a tuple of terms $\mathbf{t}$, we denote by $\mathbf{t}_\downarrow$ the tuple of constant obtained after "freezing" $\mathbf{t}$, i.e., after replacing each variable V in $\mathbf{t}$ with a new constant $c_V \in \mathbf{C}$ not occurring in $dom(D) \cup \{t, f\}$. We are now ready to give the formal reduction.

The Database D'. The new database D' is defined as follows:

$$\begin{aligned} & \{p'(\mathbf{t}, t) \mid p(\mathbf{t}) \in D\} \\ \cup\ & \{true(t), false(f), or(t,t,t), or(t,f,t), or(f,t,t), or(f,f,f)\} \\ \cup\ & \{p'(\mathbf{t}_{\downarrow}, f) \mid q \in Q \text{ and } p(\mathbf{t}) \in body(q)\}. \end{aligned}$$

The Set Σ'. The new set Σ' of DTGDs is obtained from Σ by replacing each atom of the form $p(\mathbf{X})$ occurring in a DTGD with the atom $p'(\mathbf{X}, T)$, where T is a new variable not occurring in Σ. For example, the DTGD $p(X,Y), s(Y,Z) \rightarrow \exists W p(X,W), r(W)$ will be replaced by the DTGD $p'(X,Y,T), s'(Y,Z,T) \rightarrow \exists W\, p'(X,W,T), r'(W,T)$.

The CQ q. Assume that $Q = q_1 \vee \ldots \vee q_n$. For a CQ $q_i \in Q$, we define $q_i[X_i]$ as the CQ obtained from q_i by replacing each atom of the form $p(\mathbf{t})$ with the atom $p'(\mathbf{t}, X_i)$, where X_i is a new variable not occurring in Q. The CQ q is defined as the query:

$$\exists X_1 \ldots \exists X_n \exists Y_1 \ldots \exists Y_{n+1} \left(false(Y_1) \wedge \bigwedge_{q_i \in Q} (q_i[X_i] \wedge or(Y_i, X_i, Y_{i+1})) \wedge true(Y_{n+1}) \right).$$

By construction, for each $q_i[X_i]$, there exists a homomorphism that maps $body(q_i[X_i])$ to the database D'. However, this fact does not imply that the query q is trivially entailed by $chase(D', \Sigma')$ since, in order to satisfy the atom $true(Y_{n+1})$, at least one X_i must be mapped to the constant t. Thus, by construction, the only way to entail q is to map at least one subquery $q_i[X_i]$ to $chase(D', \Sigma')$ via a homomorphism h, and also $h(X_i) = t$. Notice that the only atoms in $chase(D', \Sigma')$ containing t at the last position are the ones obtained from the original copy of D. Thus, a subquery $q_i \in Q$ is entailed by an instance $I \in chase(D, \Sigma)$ iff the corresponding instance I' of $chase(D', \Sigma')$ entails q. Since the above construction is feasible in polynomial time, the claim follows.

C. PROOFS FROM SECTION 6

C.1. Proof of Theorem 6.2

Consider a database D, a weakly-guarded set Σ of DTGDs with bounded number of body-variables and predicates of bounded arity, and an AUCQ Q. The proof is by reduction to satisfiability of GFO sentences (without constants). More precisely, we construct, in polynomial time, a database D', a set Σ' of DTGDs, and a query Q' such that $\Psi_{D',\Sigma',Q'} = (D' \wedge \Sigma' \wedge \neg Q')$ falls in GFO (without constants), the arity of the underlying schema is bounded, and $D \cup \Sigma \models Q$ iff $\Psi_{D',\Sigma',Q'}$ is unsatisfiable. Since the problem of deciding whether a GFO sentence (without constants) is satisfiable is in EXPTIME in case of predicates of bounded arity [Grädel 1999], the claim follows.

First, by exploiting the fact that Q is acyclic, we transform Q, in polynomial time, into a set Σ_Q of guarded TGDs. Instead of giving the rather tedious definition of Σ_Q, we prefer to illustrate how Σ_Q is constructed via a simple example. In what follows, we explain how a single CQ can be transformed into a set of guarded TGDs. Then, Σ_Q is obtained by transforming each disjunct of Q into a set of guarded TGDs.

Example C.1. Consider the CQ

$$q \ = \ \exists A_1 \ldots \exists A_6\, p(A_1, A_2) \wedge p(A_3, A_4) \wedge s(A_1, A_4, A_5) \wedge s(A_4, A_5, A_6).$$

In fact, this is the CQ of Example 6.1, and its hypergraph $\mathcal{H}(q)$ is shown in Figure 4. As explained in Example 6.1, the hyperedge $\{A_4, A_5, A_6\}$ is an ear due to $\{A_1, A_4, A_5\}$, and thus it can be eliminated. Moreover, $\{A_1, A_2\}$ and $\{A_3, A_4\}$ are ears due to $\{A_1, A_4, A_5\}$ and they can be eliminated. Finally, the remaining hyperedge $\{A_1, A_4, A_5\}$ is an ear, since it does not intersect with any other hyperedge, and it can be eliminated. Thus, $GYO(\mathcal{H}(q)) = (\varnothing, \varnothing)$, which means that q is acyclic. Observe that the construction of

$GYO(\mathcal{H}(q))$ defines a partial order $\prec$ over the set of hyperedges of $\mathcal{H}(q)$; in particular, $\{A_4, A_5, A_6\} \prec \{A_1, A_4, A_5\}$, $\{A_1, A_2\} \prec \{A_1, A_4, A_5\}$ and $\{A_3, A_4\} \prec \{A_1, A_4, A_5\}$. By exploiting $\prec$ we can define Σ_Q as follows:

$$\begin{aligned} s(A_4, A_5, A_6) &\rightarrow aux_1(A_4, A_5), \\ p(A_1, A_2) &\rightarrow aux_2(A_1), \\ p(A_3, A_4) &\rightarrow aux_3(A_4), \\ s(A_1, A_4, A_5), aux_1(A_4, A_5), aux_2(A_1), aux_3(A_4) &\rightarrow q^\star, \end{aligned}$$

where $q^\star$ is a an auxiliary 0-ary predicate. ∎

We proceed to rewrite, in polynomial time, D and $\Sigma \cup \Sigma_Q$ into D' and Σ', respectively, in such a way that $(D' \wedge \Sigma' \wedge \neg q^\star)$ falls in GFO (without constants), uses only predicates of bounded arity, and $D \cup \Sigma \models Q$ iff $(D' \wedge \Sigma' \wedge \neg q^\star)$ is unsatisfiable. As for Σ_Q, we prefer to illustrate how D' and Σ' are constructed via a simple example.

Example C.2. Consider the database $D = \{p(1, 2), s(1)\}$, and the set Σ of guarded TGDs (the same construction applies to weakly-guarded sets of DTGDs) consisting of

$$\begin{aligned} p(X, 2), s(X) &\rightarrow \exists Z\, r(X, Z), \\ r(1, X) &\rightarrow p(2, X). \end{aligned}$$

The database D is transformed into

$$D' = \{p_{[1,2]}, s_{[1]}\},$$

which consists only of 0-ary predicates. In other words, we encode the tuples of constants in the name of the predicates.

The set Σ' is obtained by instantiating in all the possible ways the universally quantified variables in a TGD of Σ with constants and the special symbol $\star$, which represents the fact that a variable is satisfied by a null value, and then encoding the obtained tuples in the name of the predicates. In particular, Σ' consists of the TGDs

$$\begin{aligned} p_{[1,2]}, s_{[1]} &\rightarrow \exists Z\, r_{[1,\star]}(Z), \\ p_{[2,2]}, s_{[2]} &\rightarrow \exists Z\, r_{[2,\star]}(Z), \\ p_{[\star,2]}(X), s_{[\star]}(X) &\rightarrow \exists Z\, r_{[\star,\star]}(X, Z), \\ r_{[1,1]} &\rightarrow p_{[2,1]}, \\ r_{[1,2]} &\rightarrow p_{[2,2]}, \\ r_{[1,\star]}(X) &\rightarrow p_{[2,\star]}(X). \end{aligned}$$

This completes the construction of D' and Σ'. ∎

It is not difficult to verify that the construction of $(D' \wedge \Sigma' \wedge \neg q^\star)$ is feasible in polynomial time since the DTGDs of Σ have bounded number of body-variables. Moreover, $(D' \wedge \Sigma' \wedge \neg q^\star)$ is constant-free, uses only predicates of bounded arity, and $D \cup \Sigma \models Q$ iff $(D' \wedge \Sigma' \wedge \neg q^\star)$ is unsatisfiable. However, Σ' does not immediately fall in GFO due to the fact that in the head of a DTGD an unguarded formula of the form $\exists \mathbf{Y}\, \psi(\mathbf{X}, \mathbf{Y})$, where ψ is a disjunction of conjunctions, may appear. Nevertheless, the satisfiability algorithm proposed in [Grädel 1999] is able to treat sentences which are "almost" GFO sentences as Σ', even if the existentially quantified variables are not guarded, without increasing the complexity — this is explicitly stated in a remark on page 8 of [Grädel 1999]. It is clear that we have polynomially reduced our problem to a problem in EXPTIME, and the claim follows.

C.2. Proof of Theorem 6.6

The proof is via reduction of the non-acceptance problem of an alternating linear space Turing machine M on input I. Let $M = (S, \Lambda, \delta, s_0)$ be an alternating Turing machine as defined in Section 2. For technical clarity, we adopt the assumptions made in the proof of Theorem 6.4, with the difference that now M uses exactly n tape cells, where $n = |I|$. We are going to construct a database D, a set Σ of DTGDs, and an AUCQ Q such that $D \cup \Sigma \models Q$ iff M rejects I, where $\mathsf{N}(\Sigma)$ is a fixed set of DIDs, and $sch(\mathsf{N}(\Sigma))$ consists of unary and binary predicates. By Lemma 4.7, the obtained lower bound holds even for ACQs and predicates of arity at most three. The idea of the proof is along the lines of the one of Theorem 6.4, i.e., to construct trees, which encode possible computation trees of M on the input string I, by chasing D with Σ, and then exploit Q to check their consistency. On each configuration node v, which represents the configuration C_v of M, we attach a chain of length n, which mimics the tape in C_v, and also a chain of length at most $|S|$, which encodes the state of C_v. The formal construction of D, Σ and Q follows.

The Database D. Let $D = \{conf_\exists(c)\}$, where $c \in \mathbf{C}$ is a special constant which represents the initial configuration (which is, by assumption, existential).

The Set Σ. The predicates that we are going to use are self-explanatory, and thus we proceed with the construction of Σ without describing the predicates of $sch(\Sigma)$.

— Each configuration has two successor configurations, such that a universal configuration is followed by existential configurations, and vice-versa. Note that for existential configurations, we generate two models for the successors, whereas for universal configurations, both are forced to appear in the same model:

$$conf_\exists(X) \rightarrow \exists Y\, succ_1(X,Y), conf_\forall(Y) \vee \exists Y\, succ_2(X,Y), conf_\forall(Y),$$
$$conf_\forall(X) \rightarrow \exists Y \exists Z\, succ_1(X,Y), conf_\exists(Y), succ_2(X,Z), conf_\exists(Z).$$

— Each configuration has a cell- and state-chain which simulates its tape and encodes its state, respectively:

$$conf_x(X) \rightarrow \exists Y\, cell(X,Y), \text{ for each } x \in \{\exists, \forall\},$$
$$cell(X,Y) \rightarrow \exists Z\, cell(Y,Z) \vee end(Y),$$
$$conf_x(X) \rightarrow \exists Y\, state(X,Y), \text{ for each } x \in \{\exists, \forall\},$$

$$state(X,Y) \rightarrow \exists Z\, state(Y,Z) \vee end(Y).$$

— Finally, we guess the content of each cell and the cursor position:

$$cell(X,Y) \rightarrow \bigvee_{\lambda \in \Lambda} \lambda(Y),$$
$$cell(X,Y) \rightarrow cursor(Y) \vee notCursor(Y).$$

The construction of Σ is now complete. It is easy to see that Σ does not depend on M, and also that $\mathsf{N}(\Sigma)$ is a set of DIDs.

The AUCQ Q. We now proceed with the construction of Q, which ensures that each model of $D \cup \Sigma$ encodes an invalid computation tree. Roughly, Q consists of the following disjuncts:

(1) $Q_{initial}$ for checking that in the initial configuration the tape does not contain the input string $I = a_1 \dots a_n$, the state is not s_0, and the cursor is not at the first cell;

(2) Q_{length} for checking that cell-chains are of length other than n, and state-chains are of length greater than $|S|$;
(3) Q_{cursor} for checking that more than one cells are pointed by the cursor;
(4) Q_{trans} for checking that the transition function of M is violated; and
(5) $Q_{inertia}$ for checking that the tape cells not under the cursor do not keep their old value during a transition.

We assume a fixed order on S. Given a state $s \in S$, let $\#_s$ be its position in this order. The length of the state-chain which encodes s is $\#_s$. For a predicate $p \in \{cell, state\}$, $p^i(X,Y)$ is used as a shorthand for $\exists Z_1 \ldots \exists Z_{i-1}\,(p(X,Z_1) \wedge \ldots \wedge p(Z_{i-1},Y))$. A useful subquery that we are going to use is

$$length[p]_k(X) \;\equiv\; \exists Y\left(p^k(X,Y) \wedge end(Y)\right)$$

stating that there exists a p-chain of length k. Having all the necessary ingredients in place, we are now ready to formalize the components of Q.

(1) $Q_{initial}$ is defined as (recall that $c \in \mathbf{C}$ represents the initial configuration)

$$\bigvee_{i\in[n]}\bigvee_{a\in\Lambda\setminus\{a_i\}} \exists X\left(conf_{\exists}(c) \wedge cell^i(c,X) \wedge a(X)\right) \vee \bigvee_{s\in S\setminus\{s_0\}} \left(conf_{\exists}(c) \wedge length[state]_{\#_s}(c)\right) \vee \bigvee_{1<i\leqslant n} \exists X\left(conf_{\exists}(c) \wedge cell^i(c,X) \wedge head(X)\right).$$

(2) Q_{length} is defined as

$$\bigvee_{1\leqslant i<n} \exists X\; length[cell]_i(X) \;\vee\; \exists X \exists Y\; cell^{n+1}(X,Y) \;\vee\; \exists X \exists Y\; state^{|S|+1}(X,Y).$$

(3) Q_{cursor} is defined as

$$\bigvee_{1\leqslant i<j\leqslant n} \exists X \exists Y \exists Z\left(cell^i(X,Y) \wedge cell^j(X,Z) \wedge cursor(Y) \wedge cursor(Z)\right).$$

(4) In the definition of Q_{trans}, we use the function $f : S \to \{\exists, \forall\}$, where $f(s) = \exists$ if $s \in S_\exists$; otherwise, $f(s) = \forall$. Moreover, we use the binary operator $\odot^s$, where $s \in S$, which is defined as $\vee$, if $s \in S_\exists$, and as $\wedge$, if $s \in S_\forall$. Recall that $\bar{\delta_\forall}$ and $\bar{\delta_\exists}$ denote the complement of δ relative to universal and existential, respectively; for details, see the proof of Theorem 6.4. Q_{trans} is defined as

$$\bigvee_{(s,a)\to((s_1,a_1,d_1),(s_2,a_2,d_2))\in(\bar{\delta_\forall}\cup\bar{\delta_\exists})} \;\bigodot_{i\in\{1,2\}}^{s}\; \bigvee_{j\in[n]} \exists X \exists Y \exists Z \exists V \exists W$$
$$\Big(conf_{f(s)}(X) \wedge length[state]_{\#_s}(X) \wedge cell^j(X,Y) \wedge cursor(Y) \wedge a(Y) \wedge succ_i(X,Z) \wedge$$
$$length[state]_{\#_{s_i}}(Z) \wedge cell^j(Z,V) \wedge a_i(V) \wedge cell^{j+d_i}(Z,W) \wedge cursor(W)\Big).$$

(5) Finally, $Q_{inertia}$ is defined as

$$\bigvee_{x\in\{\exists,\forall\}} \bigvee_{(a,a')\in\Lambda\times\Lambda, a\neq a'} \bigvee_{i\in[n]} \exists X \exists Y \exists Z_1 \exists Z_2\,(conf_x(X) \wedge (succ_1(X,Z) \vee succ_2(X,Z)) \wedge cell^i(X,Z_1) \wedge cell^i(Y,Z_2) \wedge notCursor(Z_1) \wedge a(Z_1) \wedge a'(Z_2)\Big).$$

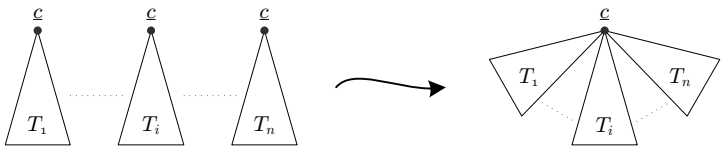

Fig. 10: Rooted tree obtained from the disjoint union of $T_1, \ldots, T_n$ after merging the root nodes.

This completes the construction of Q. It is easy to verify that each disjunct of Q is an acyclic query.

By construction, and the assumption that a rejecting configuration does not have a successor configuration, while an accepting configuration has only itself as a successor, it is not difficult to see that $D \cup \Sigma \models Q$ iff M rejects I (see the argument given in the proof of Theorem 4.5 which shows that the employed construction is correct).

D. PROOFS FROM SECTION 7

D.1. Proof of Lemma 7.3

Let $D = \{\underline{a}_1, \ldots, \underline{a}_n\}$, and $q = \exists \mathbf{X}\, p(\mathbf{X})$. Before we proceed further, we need to establish the following auxiliary claim:

CLAIM D.1. *It holds that*

$$mods(D, \Sigma) \;=\; \underbrace{\left\{ \bigcup_{i \in [n]} M_i \mid (M_1, \ldots, M_n) \in mods(\{\underline{a}_1\}, \Sigma) \times \ldots \times mods(\{\underline{a}_n\}, \Sigma) \right\}}_{\mathcal{M}}.$$

PROOF. ($\subseteq$) Fix $M \in mods(D, \Sigma)$. Clearly, for each $i \in [n]$, $M \in mods(\{\underline{a}_i\}, \Sigma)$. Thus, $(M, \ldots, M) \in mods(\{\underline{a}_1\}, \Sigma) \times \ldots \times mods(\{\underline{a}_n\}, \Sigma)$ which implies that $M \in \mathcal{M}$.

($\supseteq$) Conversely, fix a tuple $(M_1, \ldots, M_n) \in mods(\{\underline{a}_1\}, \Sigma) \times \ldots \times mods(\{\underline{a}_n\}, \Sigma)$, and let $M = \bigcup_{i \in [n]} M_i$. To show that $M \in mods(D, \Sigma)$, it suffices to show that, whenever for a DTGD $\sigma \in \Sigma$ there exists a homomorphism h such that $h(body(\sigma)) \subseteq M$, then $h(head(\sigma)) \in M$. Fix a DTGD $\sigma \in \Sigma$ and a homomorphism h such that $h(body(\sigma)) \subseteq M$. Due to the linearity of σ, there exists $i \in [n]$ such that $h(body(\sigma)) \in M_i$. But since $M_i \in mods(\{\underline{a}_i\}, \Sigma)$, $h(head(\sigma)) \in M_i$. The claim follows since $M_i \subseteq M$. □

Let us now conclude the proof of our lemma.

($\Rightarrow$) Assume that, for each $i \in [n]$, $\{\underline{a}_i\} \cup \Sigma \not\models q$, and thus there exists $M_i \in mods(\underline{a}_i, \Sigma)$ such that $M_i \not\models q$. Therefore, $(M_1 \cup \ldots \cup M_n) \not\models q$, and by the above claim we get that $D \cup \Sigma \not\models q$.

($\Leftarrow$) By hypothesis, there exists $i \in [n]$ such that $\{\underline{a}_i\} \cup \Sigma \models q$. Consequently, for each $M \in mods(\{\underline{a}_i\}, \Sigma)$, $M \models \underline{a}$. Hence, by the above claim, each instance of $mods(D, \Sigma)$ entails q, and the claim follows.

D.2. Proof of Lemma 7.6

Let $\Sigma_f = \{\rho_1, \ldots, \rho_n\}$. By definition of the tree of a model, $trees(\underline{a}, \Sigma_f)$ is isomorphic to the set

$$\mathcal{T} = \left\{ \biguplus_{i \in [n]} T_i \mid (T_1, \ldots, T_n) \in \rho_1\text{-}trees(\underline{a}, \Sigma_f) \times \ldots \times \rho_n\text{-}trees(\underline{a}, \Sigma_f) \right\},$$

where $\uplus_{i\in[n]}T_i$ is the rooted tree obtained from the disjoint union of $T_1,\ldots,T_n$ after merging the root nodes (see Figure 10). We are now ready to show our lemma.

($\Rightarrow$) Assume that for each $\rho \in \Sigma_f$ there exists $T \in \rho\text{-}trees(\underline{a},\Sigma_f)$ such that $T \not\models q$. Therefore, there exists $T' \in \mathcal{T}$ such that $T' \not\models q$. Clearly, there exists $T'' \in trees(\underline{a},\Sigma_f)$ isomorphic to T'. Assuming that $T'' = tree(M)$, for some $M \in mods(\{\underline{a}\},\Sigma_f)$, $M \not\models \underline{a}$; thus, $\{\underline{a}\} \cup \Sigma_f \not\models \underline{a}$.

($\Leftarrow$) This direction follows immediately.

D.3. Proof of Lemma 7.7

($\Rightarrow$) The claim is shown by giving a proof-tree of some $\underline{a} \in D$ and Σ which is valid w.r.t. q; assume that $q = \exists\mathbf{X}p(\mathbf{X})$. By Lemmas 7.3 and 7.6, there exist $\underline{a} \in D$ and $\rho \in \Sigma_f$ such that in every $T \in \rho\text{-}trees(\underline{a},\Sigma_f)$ there exists a path π_T from the root to a node u which is labeled by $h(p(\mathbf{X}))$, where h is a homomorphism, and the first edge e_T of π_T is labeled by a rule ρ of the form $\underline{b} \to \underline{b}_1 \vee \underline{b}_2$ or $\underline{b} \to \underline{b}_1$. Let us now construct a rooted binary tree $T' = (N, E, \lambda_1, \lambda_2)$; initially, $N = \{v\}$, $E = \varnothing$, $\lambda_1 = \{v \to \underline{a}\}$, and $\lambda_2 = \varnothing$. Clearly, $\rho\text{-}trees(\underline{a},\Sigma_f)$ can be partitioned into S_1 and S_2 such that, for each $T \in S_1$, if $e_T = (u, w)$ and h maps $\underline{b}$ into $\underline{a}$ (which is the label of u), then w is labeled by $h(\underline{b}_1)$. For each $i \in \{1, 2\}$, assume that the second node of π_T, for each $T \in S_i$, is labeled by $\underline{g}_i$. Let $N = N \cup \{w_1, w_2\}$, $E = E \cup \{(v, w_i)\}_{i\in\{1,2\}}$, $\lambda_1 = \lambda_1 \cup \{w_i \to \underline{g}_i\}_{i\in\{1,2\}}$, and $\lambda_2 = \lambda_2 \cup \{(v, w_i) \to \rho\}_{i\in\{1,2\}}$. Due to the fact that $\{\underline{g}_i\} \cup \Sigma_f \models q$, for each $i \in [2]$, Lemma 7.6 can be recursively applied finitely many times as described above, and eventually T' is constructed. The desired proof-tree is obtained from T' by replacing the skolem terms occurring in T' with distinct nulls of $\mathbf{N}$.

($\Leftarrow$) By hypothesis, there exists $\underline{a} \in D$ and a proof-tree T of $\underline{a}$ and Σ which is valid w.r.t. q. It is possible to construct from T a rooted binary tree T' such that the nodes are labeled by atoms with skolem terms (instead of nulls), the edges are labeled by rules of Σ_f, and the structural properties of T are preserved. Assume that the label of each outgoing edge of the root of T' is ρ. By Lemmas 7.3 and 7.6, it suffices to show that each ρ-tree of $\underline{a}$ and Σ_f entails q. This follows from the fact that, for each $T'' \in \rho\text{-}trees(\underline{a},\Sigma_f)$, there is a path from the root to a leaf of T' that can be mapped into T''.

D.4. Normalization Procedure

Consider a linear DTGD σ. First, we construct the set $\mathsf{N}_1(\sigma)$ by exhaustively applying the following two replacement rules, starting from σ, until each assertion is either a single-atom-head TGD, or a DTGD with a disjunction of two atoms in its head:

(1) A linear DTGD $p(\mathbf{X}) \to \bigvee_{i=1}^{n} \exists\mathbf{Y}_i\,(p_1^i(\mathbf{X},\mathbf{Y}_i),\ldots,p_{k_i}^i(\mathbf{X},\mathbf{Y}_i))$ is replaced by

$$\begin{aligned}
p(\mathbf{X}) &\to p_1^\star(\mathbf{X}) \vee p_{2..n}^\star(\mathbf{X})\\
p_1^\star(\mathbf{X}) &\to \exists\mathbf{Y}_1\, p_1^1(\mathbf{X},\mathbf{Y}_1)\\
&\vdots\\
p_1^\star(\mathbf{X}) &\to \exists\mathbf{Y}_1\, p_{k_1}^1(\mathbf{X},\mathbf{Y}_1)\\
p_{2..n}^\star(\mathbf{X}) &\to \bigvee_{i=2}^{n} \exists\mathbf{Y}_i\,(p_1^i(\mathbf{X},\mathbf{Y}_i),\ldots,p_{k_i}^i(\mathbf{X},\mathbf{Y}_i)),
\end{aligned}$$

where $p_1^\star$ and $p_{2..n}^\star$ are $|\mathbf{X}|$-ary auxiliary predicates not introduced so far.

(2) A linear TGD $p(\mathbf{X}) \rightarrow \exists \mathbf{Y}\, p_1(\mathbf{X}, \mathbf{Y}_1), \ldots, p_n(\mathbf{X}, \mathbf{Y}_n)$ is replaced by

$$\begin{aligned} p(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_1\, p_1(\mathbf{X}, \mathbf{Y}_1) \\ &\vdots \\ p(\mathbf{X}) &\rightarrow \exists \mathbf{Y}_n\, p_n(\mathbf{X}, \mathbf{Y}_n). \end{aligned}$$

Now, from $\mathsf{N}_1(\sigma)$, we obtain $\mathsf{N}(\sigma)$ by employing the construction presented in Section 2 for transforming a set of DTGDs into a set of DTGDs with only one occurrence of an existentially quantified variable. Finally, given a set Σ of DTGDs, $\mathsf{N}(\Sigma)$ is defined as $\bigcup_{\sigma \in \Sigma} \mathsf{N}(\sigma)$.

D.5. Proof of Theorem 7.15

Let D, Σ and Q be the database, the set of DIDs, and the UCQ employed in the proof of Theorem 6.4, where it is shown that ACQ answering under DIDs is 2ExpTime-hard. Our goal is to construct in polynomial time a set Σ' of multi-linear DTGDs, which depends on Σ and Q, such that $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models q$, where $q = p^\star$ with $p^\star$ being an auxiliary 0-ary predicate not occurring in $sch(\Sigma)$. In fact, our intention is to obtain Σ' by adding to Σ the TGD $\varphi \rightarrow p^\star$, for each disjunct φ of Q. However, to ensure that the added TGDs are multi-linear, we first need to modify the query Q.

Let us start by explaining why we cannot directly add the disjuncts of Q to Σ. Consider, for example, the disjunct $Q_{inertia}$

$$\bigvee_{(s,s') \in S \times S} \bigvee_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigvee_{i \in \{1,2\}} \bigvee_{j \in \{0,1\}} \exists \mathbf{X}^n \exists T \exists P \exists N_1 \exists N_2 \exists N_1' \exists N_2' \\ \left(conf[s](\mathbf{X}^n, a, 0, T, P, N_1, N_2) \wedge conf[s'](\mathbf{X}^n, a', j, N_i, T, N_1', N_2') \right),$$

which checks that the tape cells not under the cursor keep their old values during a transition. To perform such a check we just need to have access to a certain configuration and to one of its subsequent configurations. However, in an atom $conf[s](\cdot)$ we store more information than necessary, namely the previous configuration, and both subsequent configurations. This is precisely the reason why the atoms of $Q_{inertia}$ contain different variables, and its disjuncts will give rise to TGDs that are not multi-linear. Thus, we need to project out the useless variables via some (non-disjunctive) IDs, and rewrite the disjuncts of Q analogously. The projection IDs, which are part of Σ', are

$$\begin{aligned} conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_P[s](\mathbf{X}^n, C, H, T, P) \\ conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_L[s](\mathbf{X}^n, C, H, T, N_1) \\ conf[s](\mathbf{X}^n, C, H, T, P, N_1, N_2) &\rightarrow conf_R[s](\mathbf{X}^n, C, H, T, N_2). \end{aligned}$$

The subscripts P, L and R stand for previous, (next-)left and (next-)right, respectively. Now, the query $Q_{inertia}$ can be rewritten as follows:

$$\bigvee_{x \in \{L,R\}} \bigvee_{(s,s') \in S \times S} \bigvee_{(a,a') \in \Lambda \times \Lambda, a \neq a'} \bigvee_{j \in \{0,1\}} \exists \mathbf{X}^n \exists T \exists N \\ \left(conf_x[s](\mathbf{X}^n, a, 0, T, N) \wedge conf_P[s'](\mathbf{X}^n, a', j, N, T) \right).$$

Observe that, for every disjunct φ of the above query, and for every pair of atoms $\underline{a}, \underline{b}$ of φ, $var(\underline{a}) = var(\underline{b})$; thus, the TGD $\varphi \rightarrow p^\star$ is multi-linear. The same approach can be applied also for Q_{trans} and $Q_{inertia}$, without violating the multi-linearity of the added

TGDs. However, this is not true for Q_{move}. After rewriting Q_{move} as above, we get

$$\bigvee_{x\in\{L,R\}} \bigvee_{0<i,j\leqslant n} \bigvee_{(s,s')\in S\times S} \bigvee_{(c_1,c_2,c_3)\in\Lambda^3} \exists\mathbf{X}^{n-i}\exists\mathbf{Y}^{n-j}\exists T\exists N$$
$$\Bigg(\underbrace{conf_x[s](\mathbf{X}^{n-i},0,\mathbf{1}^{i-1},c_1,1,T,N)\wedge conf_P[s'](\mathbf{X}^{n-i},1,\mathbf{0}^{i-1},c_2,0,N,T)}_{\varphi_1}\wedge$$
$$\underbrace{conf_x[s](\mathbf{Y}^{n-j},1,\mathbf{0}^{j-1},c_1,1,T,N)\wedge conf_P[s'](\mathbf{Y}^{n-j},0,\mathbf{1}^{i-1},c_3,0,N,T)}_{\varphi_2}\Bigg),$$

and the TGD $\varphi_1,\varphi_2 \to p^\star$ is not multi-linear due to the variables $\mathbf{X}^{n-i}$ and $\mathbf{Y}^{n-j}$. Nevertheless, since we just need to join the variables T and N, it suffices to add to Σ the multi-linear TGDs $\varphi_1 \to aux_1(T,N)$, $\varphi_2 \to aux_2(T,N)$ and $aux_1(T,N), aux_2(T,N) \to p^\star$, where aux_1 and aux_2 are auxiliary predicates not occurring in $sch(\Sigma)$.

D.6. Proof of Theorem 7.16

The proof is via reduction of the acceptance problem of an alternating logarithmic space Turing machine M on input $I = a_1\ \ldots a_{|I|}$. Recall that logarithmic space Turing machines are equipped with a read-only input tape and a read/write work tape, where on the input tape only the cells which hold the input can be visited, while on the work tape only logarithmically many cells can be used. Let $M = (S,\Lambda,\delta,s_0)$, where $S = S_\forall \uplus S_\exists \uplus \{s_a\} \uplus \{s_r\}$ is a finite set of states partitioned into universal states, existential states, an accepting state and a rejecting state, $\Lambda = \{0,1,\sqcup\}$ is the tape alphabet with $\sqcup$ being the blank symbol, $\delta : S\times\Lambda\times\Lambda \to (S\times\Lambda\times\{-1,+1\}\times\Lambda\times\{-1,+1\})^2$ is the transition function, and $s_0 \in S$ is the initial state. We assume that M is well-behaved and never tries to read beyond its tape boundaries, and uses exactly $n = k\cdot\log|I|$ cells of the work tape, where $k > 0$.

We proceed to construct a database D, a set Σ of DIDs, where $sch(\Sigma)$ consists of unary predicates, and a CQ_1 q such that $D\cup\Sigma \models q$ iff M accepts. Each configuration of M is represented using a unary predicate of the form

$$conf[s\#a_1\ldots a_{|I|}x_1\ldots x_{|I|}\#c_1\ldots c_n y_1\ldots y_n],$$

where $s\in S$, $a_1\ldots a_{|I|}$ is the input tape, $x_1\ldots x_{|I|}\in\{0,1\}^{|I|}$ indicates the position of the cursor on the input tape (e.g., $x_1\ldots x_{i-1}x_{i+1}\ldots x_{|I|} = 0^{|I|-1}$ and $x_i = 1$ implies that the cursor is at the i-th cell), $c_1\ldots c_n\in\Lambda^n$ is the work tape, and $y_1\ldots y_n\in\{0,1\}^n$ indicates the position of the cursor on the work tape. Since the number of configurations of M on I is polynomial, we only need polynomially many predicates. We proceed with the construction of D, Σ and q.

The Database D. Let D be the database consisting of the single atom

$$conf[s_0\#a_1\ldots a_{|I|}1\overbrace{0\ldots0}^{|I|-1}\#\underbrace{\sqcup\ldots\sqcup}_{n}1\underbrace{1\ldots1}_{n}](c),$$

where c is an arbitrary constant of $\mathbf{C}$.

The Set Σ. With Σ we encode the transitions of δ. Let $f : \{0,1\}\to 2^{[|I|]}$ be the function such that $f(x)$, where $x\in\{0,1\}$, is the set of cells of the input tape which contain the

symbol x. For each transition $(s, a, b) \to ((s_1, a, +1, b_1, -1), (s_2, a, -1, b_2, +1))$ of δ, we add in Σ the following DIDs: for each $i \in f(a)$, $j \in [n-1]$ and $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n \in \Lambda^{n-1}$, $p_1(X) \to p_2(X) \vee p_3(X)$, if $s \in S_\forall$; else, $p_1(X) \to p_2(X)$ and $p_1(X) \to p_3(X)$, where

$$p_1 \;=\; \mathit{conf}[s\#a_1 \ldots a_{i-1} a a_{i+1} \ldots a_{|I|} \underbrace{0 \ldots 0}_{i-1} 1 \underbrace{0 \ldots 0}_{|I|-i} \#x_1 \ldots x_{j-1} a_1 x_{j+1} \ldots x_n \underbrace{0 \ldots 0}_{j-1} 1 \underbrace{0 \ldots 0}_{n-j}]$$

$$p_2 \;=\; \mathit{conf}[s_1\#a_1 \ldots a_{i-1} a a_{i+1} \ldots a_{|I|} \underbrace{0 \ldots 0}_{i} 1 \underbrace{0 \ldots 0}_{|I|-i-1} \#x_1 \ldots x_{j-1} b_1 x_{j+1} \ldots x_n \underbrace{0 \ldots 0}_{j-2} 1 \underbrace{0 \ldots 0}_{n-j+1}]$$

$$p_3 \;=\; \mathit{conf}[s_2\#a_1 \ldots a_{i-1} a a_{i+1} \ldots a_{|I|} \underbrace{0 \ldots 0}_{i-2} 1 \underbrace{0 \ldots 0}_{|I|-i+1} \#x_1 \ldots x_{j-1} b_2 x_{j+1} \ldots x_n \underbrace{0 \ldots 0}_{j} 1 \underbrace{0 \ldots 0}_{n-j-1}].$$

Similar DIDs are added to Σ for all the other forms of transitions occurring in δ. Finally, for each predicate of the form $\mathit{conf}[s_a \ldots]$ introduced above, which represents an accepting configuration, we add the (non-disjunctive) ID

$$\mathit{conf}[s_a \ldots](X) \to \mathit{accept}(X).$$

This completes the construction of Σ.

The CQ$_1$ q. With q we just need to ask wether an accepting configuration has been reached, i.e., $\exists X\, \mathit{accept}(X)$, where $X \in \mathbf{V}$.

By construction, $D \cup \Sigma \models q$ iff M accepts. It remains to show that the total number of DIDs that we need to construct is polynomial w.r.t. $|I|$. Recall that $n = k \cdot \log |I|$, where $k > 0$. Clearly,

$$|\Sigma| \;\leqslant\; |I| \cdot (n-1) \cdot 3^n \cdot 2 \;=\; |I| \cdot (\log |I|^k - 1) \cdot 3^{\log |I|^k} \cdot 2 \;\leqslant\; |I| \cdot (\log |I|^k - 1) \cdot |I|^{2k} \cdot 2,$$

and the claim follows.

D.7. Proof of Theorem 7.17

This result can be obtained by adapting the proof of Theorem 7.15. Since we have to simulate an alternating polynomial space (and not exponential space) Turing machine, the polynomially many cells (assume that the machine uses n cells) can be encoded in the predicates, and thus the arity becomes fixed. More precisely, we use the predicates

$$\mathit{conf}[s\#\underbrace{0 \ldots 0}_{i-1} 1 \underbrace{0 \ldots 0}_{n-i}] \qquad \text{and} \qquad \mathit{conf}_x[s\#\underbrace{0 \ldots 0}_{i-1} 1 \underbrace{0 \ldots 0}_{n-i}],$$

where $x \in \{P, L, R\}$, which correspond to the i-the cell of the encoded configuration. Now, it is straightforward to see how the set of multi-linear DTGDs given in the proof of Theorem 7.15 can be adapted, and the claim follows.

E. PROOFS FROM SECTION 8

Before giving the proofs that are missing from Section 8, let us first recall the semantics of $\mathcal{ALCHI}$. An $\mathcal{ALCHI}$-interpretation $\mathcal{I}$ is a pair $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set called *domain*, and $\cdot^\mathcal{I}$ is an *interpretation function* that maps every concept name A to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$, and every role name R to a binary relation $R^\mathcal{I}$ of $\Delta^\mathcal{I}$. $\mathcal{I}$ is

extended to complex concepts as follows:

$$
\begin{aligned}
(\neg C)^{\mathcal{I}} &:= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(B \sqcap C)^{\mathcal{I}} &:= B^{\mathcal{I}} \cap C^{\mathcal{I}} \\
(B \sqcup C)^{\mathcal{I}} &:= B^{\mathcal{I}} \cup C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{x \mid \text{ there exists } y \in \Delta^{\mathcal{I}} \text{ such that } (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\exists R^{-}.C)^{\mathcal{I}} &:= \{x \mid \text{ there exists } y \in \Delta^{\mathcal{I}} \text{ such that } (y,x) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &:= \{x \mid \text{ for all } y \in \Delta^{\mathcal{I}}, (x,y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\} \\
(\forall R^{-}.C)^{\mathcal{I}} &:= \{x \mid \text{ for all } y \in \Delta^{\mathcal{I}}, (y,x) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}.
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a model of a TBox $\mathcal{T}$ if $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for all $B \sqsubseteq C \in \mathcal{T}$. An interpretation $\mathcal{I}$ can be extended to ABoxes by demanding that $\cdot^{\mathcal{I}}$ maps every individual a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. $\mathcal{I}$ satisfies an assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and an assertion $R(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. We say that $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ if it satisfies all the assertions of $\mathcal{A}$. Given an $\mathcal{ALCHI}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $\mathcal{I}$ is a model of $\mathcal{K}$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{A}$. It is now straightforward to define the problem of query answering under $\mathcal{ALCHI}$ KBs. A CQ q over an $\mathcal{ALCHI}$ KB $\mathcal{K}$ is a conjunction of atoms of the form $A(X)$ or $R(X,Y)$, where $A \in \mathbf{N_C}$ and $R \in \mathbf{N_R}$. The answer to q is positive, written $\mathcal{K} \models q$, if $\mathcal{I} \models q$, for every model $\mathcal{I}$ of $\mathcal{K}$.

E.1. Proof of Theorems 8.1 and 8.2

Upper Bounds. The upper bounds for both theorems are obtained by reducing query answering under $\mathcal{ALCHI}$ to query answering under guarded DTGDs. First, we discuss how complex concepts can be eliminated from the given ABox by pushing them in the TBox. Such a reformulation of the given KB will allow us to consider the ABox as a relational database. Consider an $\mathcal{ALCHI}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. We construct an ABox $\mathcal{A}'$ from $\mathcal{A}$ by replacing each assertion $C(a)$, where C is a complex concept, by $C^{\star}(a)$, where $C^{\star}$ is an auxiliary concept name not occurring in $\mathcal{A}$. Then, the TBox $\mathcal{T}'$ is constructed by adding to $\mathcal{T}$ an assertion of the form $C^{\star} \sqsubseteq C$, for each complex concept C occurring in $\mathcal{A}$. It is easy to see that $(\mathcal{T}, \mathcal{A})$ and $(\mathcal{T}', \mathcal{A}')$ are equivalent w.r.t. UCQ answering. Let us now explain how an $\mathcal{ALCHI}$ TBox can be normalized in such a way that can be seen as a set of guarded DTGDs. In fact, we exploit the normalization procedure proposed in [Simancik et al. 2011] for $\mathcal{ALCH}$, i.e., $\mathcal{ALCHI}$ without inverse roles. An $\mathcal{ALCHI}$ TBox is in normal form if it only contains axioms of the form given on the left column of Table IV. It is implicit in [Simancik et al. 2011] that every $\mathcal{ALCH}$ TBox $\mathcal{T}$ can be transformed in polynomial time into an $\mathcal{ALCH}$ TBox in normal form, which is equivalent to $\mathcal{T}$ w.r.t. UCQ answering. This result can be straightforwardly extended to $\mathcal{ALCHI}$. From the above discussion, we immediately get the following auxiliary result:

LEMMA E.1. *Consider an $\mathcal{ALCHI}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. We can construct in polynomial time an $\mathcal{ALCHI}$ KB $\mathcal{K}' = (\mathcal{T}', \mathcal{A}')$ such that $\mathcal{A}'$ contains only concept and role names, and $\mathcal{K} \models Q$ iff $\mathcal{K}' \models Q$, for every UCQ Q.*

As shown in Table IV, given an $\mathcal{ALCHI}$ TBox $\mathcal{T}$ in normal form, every axiom of $\mathcal{T}$ can be equivalently rewritten as a first-order sentence, which is either a guarded DTGD or a constraint of the form $\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow \bot)$, where $\bot$ denotes the truth constant *false*, which may lead to an inconsistency. Such inconsistencies can be encoded in the query by considering the left-hand side of the constraints as disjuncts of the given UCQ. Moreover, observe that these disjuncts will be BTWCQs and ACQs, and thus, if the given UCQ is of bounded treewidth (resp., acyclic), then the obtained UCQ remains of bounded treewidth (resp., acyclic). Now, an ABox that contains only

$\mathcal{ALCHI}$ Axiom	First-Order Representation
$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq \bot$	$\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow \bot)$
$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B_1 \sqcup \ldots \sqcup B_m$	$\forall X(A_1(X) \wedge \ldots \wedge A_n(X) \rightarrow B_1(X) \vee \ldots \vee B_m(X))$
$A \sqsubseteq \exists R.B$	$\forall X(A(X) \rightarrow \exists Y\, R(X,Y) \wedge B(Y))$
$A \sqsubseteq \exists R^-.B$	$\forall X(A(X) \rightarrow \exists Y\, R(Y,X) \wedge B(Y))$
$A \sqsubseteq \forall R.B$, $\exists R^-.A \sqsubseteq B$	$\forall X(A(X) \wedge R(X,Y) \rightarrow B(Y))$
$A \sqsubseteq \forall R^-.B$, $\exists R.A \sqsubseteq B$	$\forall X(A(X) \wedge R(Y,X) \rightarrow B(Y))$
$R \sqsubseteq S$, $R^- \sqsubseteq S^-$	$\forall X \forall Y(R(X,Y) \rightarrow S(X,Y))$
$R \sqsubseteq S^-$, $R^- \sqsubseteq S$	$\forall X \forall Y(R(X,Y) \rightarrow S(Y,X))$

Table IV: Normal form and first-order representation.

concept and role names (and not complex concepts) can be naturally seen as a relational database. Therefore, from Lemma E.1 and the above discussion, we conclude the following: given an $\mathcal{ALCHI}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a UCQ (resp., BTWUCQ, AUCQ) Q, we can construct in polynomial time a database $D_{\mathcal{A}}$, a set $\Sigma_{\mathcal{T}}$ of guarded DTGDs, and a UCQ (resp., BTWUCQ, AUCQ) $Q_{\mathcal{T}}$ such that $\mathcal{K} \models Q$ iff $D_{\mathcal{A}} \cup \Sigma_{\mathcal{T}} \models Q_{\mathcal{T}}$, and the next result follows:

PROPOSITION E.2. *UCQ (resp., BTWUCQ, AUCQ) answering under $\mathcal{ALCHI}$ KBs can be reduced to UCQ (resp., BTWUCQ, AUCQ) answering under guarded DTGDs in polynomial time.*

Clearly, the above proposition, combined with our results on query answering under guarded DTGDs, implies the upper bounds stated in Theorems 8.1 and 8.2.

Lower Bounds. The desired lower bounds are inherited from Theorems 4.6, 5.1 and 6.6. In fact, the sets of DIDs employed in the proofs of the above results can be equivalently rewritten as DL axioms.

We first focus on the proof of Theorems 4.6 and 5.1; recall that the above theorems have exactly the same proof. The DIDs employed in this proof have one of the following forms that correspond to a DL axiom:

$$\begin{aligned} A(X) \rightarrow B_1(X) \vee \ldots \vee B_n(X) &\equiv A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n \\ A(X) \rightarrow \exists Y\, R(X,Y) &\equiv A \sqsubseteq \exists R.\top \\ R(X,Y) \rightarrow A(Y) &\equiv \exists R^-.\top \sqsubseteq A \\ R(X,Y) \rightarrow S(X,Y) &\equiv R \sqsubseteq S \\ R(X,Y) \rightarrow S(Y,X) &\equiv R^- \sqsubseteq S. \end{aligned}$$

Clearly, an axiom of the form $A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n$ can be rewritten as

$$\begin{aligned} A &\sqsubseteq B_1 \sqcup B_{2..n} \\ B_{2..n} &\sqsubseteq B_2 \sqcup B_{3..n} \\ &\vdots \\ B_{(n-2)..n} &\sqsubseteq B_{n-2} \sqcup B_{(n-1)n} \\ B_{(n-1)n} &\sqsubseteq B_{n-1} \sqcup B_n, \end{aligned}$$

and the next result follows:

PROPOSITION E.3. *Consider a DL $\mathcal{L}$ that is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^-.\top \sqsubseteq A \qquad R \sqsubseteq S \qquad R^- \sqsubseteq S,$$

*where A, A_1, A_2, A_3 are concept names and R, S are role names. Then, (BTW)UCQ answering under $\mathcal{L}$ is 2*EXPTIME*-hard in combined complexity.*

Let us now focus on the proof of Theorem 6.6. The DIDs employed in this proof have one of the following forms that correspond to DL axioms (possibly more than one):

$$\begin{aligned}
A(X) \rightarrow \exists Y\, R(X,Y) &\equiv A \sqsubseteq \exists R.\top \\
R(X,Y) \rightarrow A(Y) &\equiv \exists R^{-}.\top \sqsubseteq A \\
R(X,Y) \rightarrow \exists Z\, S(Y,Z) \vee A(Y) &\equiv \exists R^{-}.\top \sqsubseteq B_1 \quad B_1 \sqsubseteq B_2 \sqcup A \quad B_2 \sqsubseteq \exists S.\top \\
R(X,Y) \rightarrow A_1(Y) \vee A_2(Y) \vee A_3(Y) &\equiv \exists R^{-}.\top \sqsubseteq B \quad B \sqsubseteq A_1 \sqcup A_{23} \quad A_{23} \sqsubseteq A_2 \sqcup A_3,
\end{aligned}$$

and the next result follows:

PROPOSITION E.4. *Consider a DL $\mathcal{L}$ that is powerful enough for expressing the following inclusion assertions:*

$$A_1 \sqsubseteq A_2 \sqcup A_3 \qquad A \sqsubseteq \exists R.\top \qquad \exists R^{-}.\top \sqsubseteq A,$$

where A, A_1, A_2, A_3 are concept names. Then, AUCQ answering under $\mathcal{L}$ is EXPTIME*-hard in combined complexity, even when the TBox is fixed.*